Programação Coindutiva

Cálculos e Aplicações

Paula Cristina Riobom Soares Ribeiro

Departamento de Matemática
Escola de Ciências
Universidade do Minho
2005

Dissertação submetida à Universidade do Minho para obtenção do grau de Mestre em Matemática Computacional.

É autorizada a reprodução integral desta tese, apenas para efeitos de investigação.

Resumo

Como estruturas formais, tanto as álgebras *iniciais* como as coalgebras *finais* fornecem descrições abstractas de uma variedade de fenómenos em programação, em particular, das estruturas de *dados* e dos *comportamentos*, respectivamente. As propriedades universais que as definem oferecem métodos de definição e princípios de prova, *i.e.*, uma base para o desenvolvimento de cálculos de programas directamente baseados em (em rigor, orientados por) especificações de tipos. Mais ainda, tais propriedades universais podem ser codificadas em *combinadores* e usadas, não apenas para calcular programas, mas também para programar. Na programação funcional o papel destes combinadores tornou-se fundamental como base de toda uma disciplina de derivação e de transformação de algoritmos. Do lado coalgébrico, *modelar coalgebricamente* sistemas dinâmicos e raciocinar por *coindução* tem emergido recentemente como uma área de pesquisa activa.

Neste contexto, a actual dissertação tem como principal objectivo o estudo de estruturas *coalgébricas* e a sua aplicação à construção de programas. A ênfase é colocada nos princípios que permitem raciocinar sobre tais estruturas, conduzindo a uma aproximação por *cálculo* à coindução que evita a construção explícita das bissimulações. A aproximação é discutida no contexto de dois casos de estudo: um em torno do cálculo de sequências infinitas e outro da especificação coindutiva da álgebras de processos clássicas. Todas as construções e exemplos apresentados são prototipados na linguagem funcional HASKELL.

Abstract

As formal structures, both *initial* algebras and *final* coalgebras provide abstract descriptions of a variety of phenomena in programming, in particular of *data* and *behavioural* structures, respectively. Being defined by universal properties, both entail definitional and proof principles, *i.e.*, a basis for the development of program calculi directly based on (actually driven by) type specifications. Moreover, such properties can be turned into programming *combinators* and used, not only to calculate programs, but also to program with. In functional programming the role of such universals has been fundamental to a whole discipline of algorithm derivation and transformation. On the coalgebraic side, *coalgebraic modelling* of dynamical systems and reasoning by *coinduction* has recently emerged as an active area of research.

Such is the context of the present dissertation: the study of *coalgebraic* structures and their application to systems' construction. Its main focus, however, is placed on reasoning principles for such structures, introducing an entirely *calculational* approach to coinduction which avoids the explicit construction of bisimulations, and, therefore, promotes a reasoning style closer to the actual program construction practice. The approach is discussed in the context of two case-studies: a calculus of infinite sequences and a coinductive formulation of classical process algebras. All constructions and examples presented are prototyped in the functional language HASKELL.

Agradecimentos

Este espaço é dedicado àqueles que deram a sua contribuição para que esta dissertação fosse realizada. A todos eles deixo aqui o meu agradecimento sincero.

Em primeiro lugar agradeço ao Prof. Doutor Luís Soares Barbosa a forma como me orientou desde a pesquisa de informação à concretização das ideias que iam surgindo. As notas dominantes da sua orientação foram a utilidade das suas recomendações e artigos sugeridos e a forma como se mostrou paciente e generoso nos momentos *stop and go* que esta dissertação foi "vivendo".

Em segundo lugar, agradeço ao Nuno Rodrigues por me ter proporcionado "ver" os processos.

Por fim, deixo uma palavra de agradecimento aos professores do MMC pela forma como leccionaram, permitindo-me chegar aqui. Devo também agradecer aos colegas de grupo pelos almoços de sexta-feira, em que tinham lugar o incentivo, e as sessões de estudo e dúvidas colectivas, em particular, à Ana Luísa Nunes pela disposição com que realizámos os muitos trabalhos em comum.

Índice

1 Introdução **3**
- 1.1 Contexto . . . 3
- 1.2 Coalgebras e Modelos Coindutivos . . . 5
- 1.3 Raciocínio por Cálculo . . . 7
- 1.4 Objectivos e Estrutura da Dissertação . . . 12

2 Fundamentos **15**
- 2.1 Introdução . . . 15
- 2.2 Categorias e Propriedades Universais . . . 16
- 2.3 Coalgebras e Coindução . . . 22
- 2.4 Cálculo de Relações Binárias . . . 26

3 Estudo de Caso: Sequências Infinitas **29**
- 3.1 Definição e Prova por Coindução . . . 29
- 3.2 Anamorfismo . . . 33
  - 3.2.1 Codificação em HASKELL . . . 34
  - 3.2.2 Definição Coindutiva . . . 35
  - 3.2.3 Provas por Coindução . . . 43
  - 3.2.4 Um Exercício sobre $\mathbb{R}^{\omega}$ . . . 57
- 3.3 Apomorfismo . . . 62
  - 3.3.1 Codificação em HASKELL . . . 65
- 3.4 Futumorfismo . . . 66
  - 3.4.1 Codificação em HASKELL . . . 72

4 Estudo de Caso: Álgebra de Processos **75**
- 4.1 Introdução . . . 75
- 4.2 Especificação Coindutiva de Álgebras de Processos . . . 78
- 4.3 Animação Funcional . . . 83
- 4.4 Novos Combinadores de Processsos . . . 90
  - 4.4.1 Internalização . . . 90
  - 4.4.2 Interrupção . . . 93

4.4.3 Leis de Fusão Condicional 96
4.4.4 Recuperação 99
4.5 Transposição Relacional 101
4.5.1 Coalgebras e Sistemas de Transição 101
4.5.2 Simulação e Bissimulação 105
4.6 Uma Caracterização da Equivalência Fraca 110
4.6.1 Bissimulação Fraca 110
4.6.2 Exemplos 116

5 Conclusões e Trabalho Futuro **123**
5.1 Trabalho Futuro 124

A Coindução nas Streams **127**

B Animador para Cálculos de Processos **131**

Capítulo 1

Introdução

Resumo

Neste capítulo apresenta-se o caminho que foi seguido na construção da dissertação, identificando-se o seu contexto e objectivos.

"É surpreendente. É simplesmente surpreendente que a matemática seja tão poderosa para formular as leis físicas. Terá que ser assim? Não sei. Mas até hoje parece que tem de ser a matemática a descrever o mundo... Não temos outro instrumento que o consiga fazer."
Brian Greene

1.1 Contexto

Na sociedade em que vivemos, no quotidiano, seja por trabalho ou por lazer, cada vez mais somos dependentes do computador e consequentemente, do correcto funcionamento dos programas. Os computadores não só asseguram o funcionamento de diversos sistemas vitais na ordem económica e social, como expandiram a fronteiras não imaginadas a capacidade de resolução de problemas e tomada de decisão das sociedades. E, no entanto, a computação é talvez a única área de saber em que uma tecnologia popular e efectiva emergiu antes que uma metodologia científica e fundamentos matemáticos adequados tivessem sido desenvolvidos.

É bem sabido, por outro lado, que as disciplinas de Ciências Aplicadas e Engenharia evoluem precisamente na medida em que se baseiam em sólidos resultados científicos, que, por sua vez, recorrem à Matemática para poderem ser *rigorosamente expressos* e *susceptívies de raciocínio*. O mesmo se passa, evidentemente, nas Ciências da Computação. Não lidam estas com a explicação de fenómonos naturais nem com a construção de artefactos (pontes, automóveis ou cidades), mas com sistemas complexos que gerem, estruturam e transformam um dos mais importantes recursos da Humanidade — a informação e o conhecimento. Os programas que realizam estes sistemas complexos e interactuantes podem ser vistos como textos matemáticos com significado formal preciso, 'codificando' propriedades sobre o sistema que implementam. Programar não é mais do que resolver um problema e um *algoritmo*[1] constitui um modelo no sentido exacto que a palavra tem, por exemplo, em Física ou Engenharia.

Assim, as Ciências da Computação, nas quais este trabalho se insere, são essencialmente um campo privilegiado para a modelação matemática e a resolução de problemas. Recorde-se que a estratégia para a resolução de problemas tem uma receita antiga que, com origem na Física, se corporizou no método científico que é hoje parte da literacia científica dos nossos contemporâneos. O método de resolução de problemas, na formulação dada por Polya [Pol45], em 1945, baseia-se nas etapas seguintes:

- *Interpretar* o enunciado, explicitando os dados e o objectivo do problema. Usar condições matemáticas para traduzir os dados quando tal for adequado.

- *Estabelecer* e *executar* um plano de resolução do problema, usando tabelas, esquemas, decidindo sobre o uso do cálculo mental, de algoritmo de papel e lápis, criando versões mais simples do problema dado, na procura de leis de formação, conforme o tipo de situação.

- *Verificar* se o plano se adequa ao problema, tomando as decisões adequadas ao resultado da verificação, nomeadamente interpretando em contexto as soluções e decidindo sobre a sua razoabilidade.

Subjacente a esta estratégia estão dois conceitos fundamentais, que informam toda esta dissertação:

- *Modelação*, entendida como a capacidade de escolher as abstracções adequadas a cada problema.

[1] Algoritmo vem do mesmo radical que *algarismo* e este vem da repetição do nome do matemático persa Al-Khwarismi, que escreveu um livro sobre álgebra.

- *Cálculo*, no sentido que tais abstracções só são úteis se forem expressas num contexto matemático suficientemente rico para permitir estabelecer as propriedades dos modelos e transformá-los.

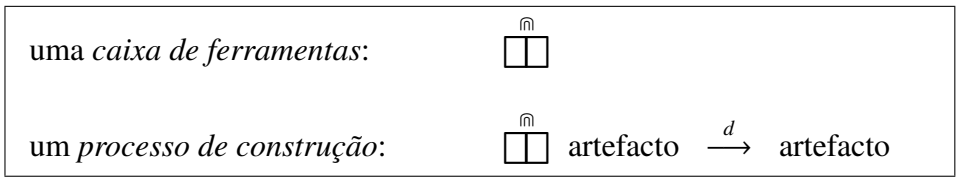
Assim, a investigação em Ciências da Computação estrutura-se tipicamente em torno de duas questões:

- Quais as abstracções (estruturas, modelos) mais adequadas para representar um determinado sistema, problema ou domínio de aplicação?
- De que forma tais abstrações tornam possível o *raciocínio efectivo* sobre a construção dos algoritmos e o desenvolvimento dos programas?

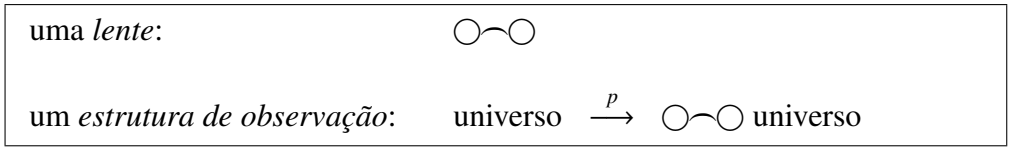
Nesta dissertação iremos centrar-nos num tipo particular de *modelos* e num determinado *estilo de raciocínio*. Os modelos serão *coalgébricos*. O estilo de raciocínio será uma versão *calculacional* da *coindução*. As duas secções que se seguem procuram motivar o leitor para estes dois tópicos a que dedicamos o nosso estudo e são centrais neste trabalho.

1.2 Coalgebras e Modelos Coindutivos

Um dos modelos mais elementares de um processo computacional é o de uma *função* $f : I \longrightarrow O$ que especifica uma regra de transformação entre duas estruturas I e O. O seu comportamento é totalmente captado pelos resultados que produz que, por sua vez, são univocamente determinados pelos argumentos. Com ela podemos tanto *construir* estruturas como *observá-las*. Formalmente, a dimensão construtiva é essencialmente *algébrica*; a segunda é, de forma dual, *coalgébrica*. No primeiro caso a função desempenha o papel de uma *caixa de ferramentas*



No segundo, o papel de uma *lente*



Como veremos com mais detalhe no capítulo 2, a interpretação matemática do *processo de construção* e da *estrutura de observação* acima referidas, é a de, respectivamente, *álgebra* e *coalgebra*. Do mesmo modo, a *caixa de ferramentas* e a *lente* representam a assinatura de operações disponíveis de construção e observação, respectivamente. Tecnicamente, na linguagem da teoria das categorias que também revisitaremos no capítulo 2, são *functores*.

É bem conhecido o facto de as álgebras (*iniciais*) fornecerem modelos abstractos para as estruturas de dados usadas no desenvolvimento de sistemas computacionais. Enquanto propriedade universal, a *inicialidade* comporta quer um método de definição quer um princípio de prova que, em conjunto, constitui a base para o desenvolvimento de cálculos de programas directamente baseados (em rigor, orientados) por especificações de tipos.

Mais ainda, propriedades universais podem ser codificadas em *combinadores* de programas e utilizadas, não apenas para calcular programas, mas também como estruturas de programação. Na programação funcional [Bir98], em particular, o papel destes combinadores, em conjunção com o raciocínio típico em teoria das categorias, tornou-se fundamental como base de toda uma disciplina de derivação e transformação de algoritmos. Tal disciplina tem origem no chamado *formalismo de Bird-Meertens* [Bir87, BM87] e no trabalho fundacional de T. Hagino [Hag87]. Desde então esta área das Ciências da Computação conheceu um notável progresso, testemunhado pela vasta bibliografia publicada quer ao nível da teoria quer das aplicações — ver [Mal90, MFP91, BM97], entre muitas outras referências.

De forma similar, mas formalmente dual, as coalgebras (*finais*) conduzem a caracterizações abstractas de fenómonos *comportamentais*, *i.e.*, de estruturas que não são caracterizadas por conjuntos de *construtores* (especificando o processo de 'construção' de um 'objecto'), mas antes por um conjunto de *observadores* que, em conjunto, especificam aquilo que pode ser observado do estado de um sistema ao longo da sua evolução temporal. Tais estruturas, que podem ser *observadas* nos seus efeitos mas não construídas nem controladas arbitrariamente, são ditas *coindutivas*. Exemplos destas estruturas abundam nas Ciências da Computação: autómatos, sistemas de transição etiquetados (tipicamente usados em semântica operacional), processos comunicantes, estruturas de dados infinitas, etc.

Um exemplo típico para ilustrar a dualidade entre o ponto de vista da álgebra inicial e o da colagebra final é dado por uma das estruturas mais utilizadas em com-

putação: a noção de *sequência* de valores. Sequências finitas são *construídas* por duas operações básicas: a constante que representa a sequência vazia e a operação que acrescenta um novo elemento à cabeça de uma sequência dada. A álgebra inicial, *i.e.*, dos termos gerada por estas duas operações representa, como facilmente se verifica, todas as possíveis sequências finitas sobre um dado domínio de valores. As sequências *infinitas*, no entanto, não podem ser construídas por recurso a qualquer procedimento algoritmíco razoável, *i.e.*, finitário. Em rigor, podem apenas ser *observadas* através de duas funções que retornam, respectivamente, a sua cabeça e cauda, esta última ainda uma sequência infinita.

A investigação em teoria das coalgebras e coindução tem vindo a conquistar um lugar relevante ao longo dos últimos 10 anos, em particular nas suas aplicações às Ciências da Computação. Isso mesmo pode ser conferido analisando as actas da série de workshops internacionais em *Coalgebraic Methods in Computer Science*, iniciadas em 1998. As referências [JR97, Rut00, Gum99] assim como algumas teses recentes (por exemplo, [Len98, Ven00, Kur01]) oferecem uma visão detalhada da área e suas aplicações.

1.3 Raciocínio por Cálculo

O estilo de raciocínio mais popular em Matemática, que se pode sumariar no slogan *teorema-seguido-de-prova*, revela-se em muitas situações pouco adequado para a modelação de algoritmos e programas. A razão é que tal estilo supõe uma aproximação à resolução de problemas em que o sistema é primeiro construído e, de seguida, verificada a sua correcção e/ou adequação ao fim em vista.

A atitude alternativa é a que, em vez de construir o sistema para posteriormente o validar face ao modelo, favorece o seu *cálculo* através da derivação formal de toda uma cadeia de modelos progressivamente mais concretos. Diversos autores têm discutido esta dicotomia entre estilos de raciocínio *orientados à verificação* ou *dirigidos por cálculo* (veja-se, por exemplo, [GS93, Zei99, Bac03], entre outros).

A preocupação por raciocinar sobre os programas coindutivos de uma forma *calculacional*, explorando directamente as propriedades *universais* das construções utilizadas, é uma trave-mestra desta dissertação. Antes de prosseguirmos, porém, tentemos clarificar a dicotomia referida através de alguns exemplos da Matemática corrente.

Suponhamos, como primeiro exemplo, que pretendemos indagar se $ln(2) + ln(7)$ é

maior ou menor que $ln(3)+ln(5)$. O estilo associado à *verificação* procede por suposição e prova. Com alguma clarividência assumimos que a primeira expressão é menor que a segunda e induzimos argumentos que sustentem essa hipótese. Assim,

(1)	se $a > 1$ então log_a é estritamente crescente	
(2)	se $a < b$ então $ln(a) < ln(b)$	por $e > 1$ e (1)
(3)	$log_a(xy) = log_a(x) + log_a(y)$	
(4)	$14 < 15$	
(5)	$14 = 2 \times 7$ e $15 = 3 \times 5$	
(6)	$ln(14) < ln(15)$	por (2) e (4)
(7)	$ln(2) + ln(7) < ln(3) + ln(5)$	por (3), (5) e (6)

A prova lida com factos simples e a sua coerência lógica é indiscutível. No entanto, é difícil de reproduzir ou reutilizar (e, por isso mesmo, de ensinar), na medida em que pouco contribui para o efectivo entendimento do problema.

A sua resolução *por cálculo* é totalmente diversa. Não parte de uma hipótese totalmente construída, mas da identificação de uma incógnita. Essa identificação vai guiar o desenvolvimento do cálculo através da aplicação de leis adequadas que conduzem à instanciação da incógnita. Neste exemplo a incógnita é a relação de ordem entre as duas expressões. Assim,

$$
\begin{array}{ll}
 & ln(2) + ln(7) \ \square \ ln(3) + ln(5) \\
\equiv & \quad \{ \text{ pelo logaritmo do produto } \} \\
 & ln(2 \times 7) \ \square \ ln(3 \times 5) \\
\equiv & \quad \{ \text{ aritmética} \} \\
 & ln(14) \ \square \ ln(15) \\
\equiv & \quad \{ \text{ como } 14 < 15 \text{ e } ln\ x \text{ é crescente} \} \\
 & \square \quad \text{é} \quad <
\end{array}
$$

Esta é uma prova por construção que, como tal, envolve manipulação sintáctica e por leis gerais. O objectivo é claro e guia todo o desenvolvimento: determinar a relação de ordem entre $ln(2) + ln(7)$ e $ln(3) + ln(5)$.

Vejamos, agora, um outro exemplo, tirado da trigonometria. Primeiro a prova por verificação:

(1) $\sin x = \cos x$ para $x = \frac{\pi}{4} + k\pi, k \in Z$
(2) $\sin(-x) = -\sin x, \forall x \in \mathbb{R}$
(3) $\sin x = \sin \alpha \equiv x = \alpha + 2k\pi \vee x = (\pi - \alpha) + 2k\pi, k \in Z$
(4) $-\frac{3}{4}\pi = \frac{\pi}{4} - \pi$
(5) $\cos\left(-\frac{3}{4}\pi\right) = \sin\left(-\frac{3}{4}\pi\right)$ por (1) e (4)
(6) $-\frac{3}{4}\pi = \pi - \left(-\frac{\pi}{4}\right)$
(7) $\sin\left(-\frac{3}{4}\pi\right) = \sin\left(-\frac{\pi}{4}\right)$ por (3) e (6)
(8) $\sin\left(-\frac{\pi}{4}\right) = -\sin\left(\frac{\pi}{4}\right)$ por (2)
(9) $\sin\left(\frac{\pi}{4}\right) + \cos\left(-\frac{3}{4}\pi\right) = \sin\frac{\pi}{4} - \sin\frac{\pi}{4} = 0$ por (7), (8) e aritmética

De novo esta prova é fácil de verificar, mas pobre na produção de intuição matemática e não tão fácil de voltar a repetir. Vejamos, agora, a prova *por cálculo*:

$$
\begin{array}{ll}
 & \sin\frac{\pi}{4} + \cos\left(-\frac{3}{4}\pi\right) \ \square \ 0 \\
\equiv & \quad \{ \text{ por (1) } \} \\
 & \sin\frac{\pi}{4} + \sin\left(-\frac{3}{4}\pi\right) \ \square \ 0 \\
\equiv & \quad \{ \text{ por (2) e (4) } \} \\
 & \sin\frac{\pi}{4} + \sin\left(-\frac{\pi}{4}\right) \ \square \ 0 \\
\equiv & \quad \{ \text{ por (2) } \} \\
 & \sin\frac{\pi}{4} - \sin\frac{\pi}{4} \ \square \ 0 \\
\equiv & \quad \{ \text{ aritmética } \} \\
 & \square \quad \text{é} \quad =
\end{array}
$$

Estes exemplos ilustram o tipo de raciocínio que em Ciências da Computação se supõe mais adequado para o desenvolvimento e transformação de programas. Estes são calculados por um processo em que a *correcção* é garantida por *construção*. O leitor é referido a [Gib02, BM97] para uma abordagem detalhada dos processos de construção formal de software.

Na prática o sucesso do estilo calculacional é condicionado pela existência de *notações* adequadas. De facto, o cálculo requer notações que sejam *genéricas, concisas e exactas* [Bac03], ou, numa palavra, *elegantes*, no sentido dado por E. W.

Dijsktra: *simple and remarkably effective* [DS90]. O uso excessivo de quantificadores numa fórmula, por exemplo, pode ser olhado como uma descrição intuitiva de um problema, mas torna a sua manipulação formal muito difícil.

Em verdade, uma boa parte da História da Matemática é a história do desenvolvimento de notações progressivamente mais simples, concisas e adequadas ao cálculo. Por exemplo, desde os primórdios da Humanidade, o Homem procura estabelecer "correspondências biunívocas" e encontrar formas de as representar. Quando o Homem passou de actividades primitivas exclusivas de caça e de pesca às de pastagem e de agricultura, terá sentido a necessidade de contar com maior precisão e, por isso, talvez se possa afirmar que aí surgiu o primeiro sistema de *matemática aplicada*. A notação era rudimentar: um corte numa casca de árvore por cada cabeça de gado. Assim surgiu, o que mais tarde os matemáticos definiram como *correspondência biunívoca entre duas ordens*: tantos cortes como tantos animais.

Depois, a Matemática, tal como qualquer outra ciência, foi evoluindo e simplificando a sua forma de escrita. Vejamos o seguinte problema que aparece numa tábua do antigo período da Babilónia com vinte e um problemas (BM 13901):

"Adicionei a superfície, o lado e o terço do lado do meu quadrado: 0;55. Colocarás 1. Juntarás um terço: 1;20. Quadrarás a sua metade, 0;40: 0;26,40. Juntarás a 0;55: 1;21,40. É o quadrado de 1;10. Subtrairás 0;40, que tinhas quadrado, de 1;10: 0;30 é o lado do quadrado."

Traduzindo este enunciado para a linguagem aritmética-algébrica actual e a sua respeciva resolução vem:

$$l^2 + l + \tfrac{1}{3}l = \tfrac{11}{12}$$

A resolução será:

$$\begin{aligned} 1 + \frac{1}{3} &= \frac{4}{3} \\ \left(\frac{4}{3} \times \frac{1}{2}\right)^2 &= \left(\frac{2}{3}\right)^2 = \frac{4}{9} \\ \frac{4}{9} + \frac{11}{12} &= \frac{49}{36} = \left(\frac{7}{6}\right)^2 \\ \frac{7}{6} - \frac{4}{6} &= \frac{1}{2} \end{aligned}$$

A salientar que $0;55$, uma vez que se trabalhava com o sistema sexagesimal, é $\frac{55}{60} = \frac{5\times11}{5\times12} = \frac{11}{12}$. Do mesmo modo se procede para os outros números apresentados no sistema sexagesimal.

Esta tendência para a economia notacional surge como uma espécie de implosão da linguagem. O objectivo é sempre o mesmo: facilitar a manipulação das fórmulas, facilitar o exercício do cálculo. Como outro exemplo, contraste-se a fórmula

$$.60.\tilde{p}.2.ce \;\; son \; yguales \; a \;\; .30.co$$

usada por Pedro Nunes no século XVI no seu *Libro de Algebra* (Coimbra, 1567), que actualmente se escreve como $60 + 2x^2 = 30x$.

Mas afinal qual a transformação notacional adequada ao cálculo de programas? A resposta que, na senda de [BM87, DS90, BM97], vamos explorar nesta dissertação é muito simples: *evitar as variáveis*. Em particular, a álgebra das funções (ou das relações), generalizada na teoria das categorias, substitui a aplicação de funções a argumentos, pela composição, ao mesmo tempo que procura definições em termos de propriedades genéricas e não de representações *ad hoc*. Um pequeno exemplo ajudará a ilustrar este ponto.

Considere-se a seguinte função de *emparelhamento*:

$$\langle f, g\rangle\, x \;=\; (f\, x, g\, x)$$

e tentemos mostrar o facto elementar de que qualquer função que constrói um par ordenado é uma função deste tipo, *i.e.*, que

$$\langle \pi_1 \cdot h, \pi_2 \cdot h\rangle \;=\; h$$

onde π_1 e π_2 são funções que selecionam o primeiro e o segundo elemento de um par ordenado, respectivamente. A definição parece bastante óbvia, mas não é fácil de manipular. A nossa tentativa de cálculo começa por supor que $h\, x = (b, c)$. Então,

$$\begin{array}{ll} & \langle \pi_1 \cdot h, \pi_2 \cdot h\rangle\, x \\ = & \quad \{ \text{ definição de função de emparelhamento, composição } \} \\ & \langle \pi_1(hx), \pi_2(hx)\rangle \\ = & \quad \{ \text{ definição de } h \} \\ & \langle \pi_1(b, c), \pi_2(b, c)\rangle \\ = & \quad \{ \text{ definição de } \pi_1 \text{ e } \pi_2 \} \\ & (b, c) \\ = & \quad \{ \text{ definição de } h \} \\ & hx \end{array}$$

Em alternativa, consideremos uma definição dita *pointfree* (*i.e.*, que não recorre a representações explícitas de valores) da mesma função de emparelhamento. Tal definição é *implícita* (mas, como veremos no capítulo 2, *universal*): $\langle f, g\rangle$ é a *solução única* das equações

$$\pi_1 \cdot x = f \quad \text{e} \quad \pi_2 \cdot x = g$$

isto é,

$$k = \langle f, g\rangle \quad \Leftrightarrow \quad \pi_1 \cdot k = f \ \wedge \ \pi_2 \cdot k = g \tag{1.1}$$

Atentemos no que ela caracteriza: se a função k é o emparelhamento de f e g, então, quando composta com π_1 (respectivamente, π_2) reduz a f (respectivamente, a g).

Com esta definição a prova do nosso resultado torna-se trivial:

$$\begin{array}{ll} & h = \langle \pi_1 \cdot h, \pi_2 \cdot h\rangle \\ \equiv & \quad \{ \text{ por (1.1) com } f = \pi_1 \cdot h,\ g = \pi_1 \cdot h \} \\ & \pi_1 \cdot h = \pi_1 \cdot h \ \wedge \ \pi_2 \cdot h = \pi_2 \cdot h \end{array}$$

É exactamente esta simplicidade de cálculo que procuramos para raciocionar sobre programas e os seus modelos.

1.4 Objectivos e Estrutura da Dissertação

No contexto delineado nas secções anteriores, os objectivos do trabalho que conduziram a esta dissertação foram os seguintes:

1. Estudar os fundamentos matemáticos do cálculo de programas funcionais na sua vertente coindutiva, identificando, a partir da literatura, os esquemas de definição e prova por coindução, e aplicando-os a situações concretas.

2. Comparar o estilo de raciocínio coindutivo por cálculo e o tipo de provas por exibição de bissimulação mais comuns na literatura.

3. Desenvolver aplicações desses esquemas e desse tipo de raciocínio em determinadas áreas de investigação relevantes para as Ciências da Computação.

4. Construir uma biblioteca na linguagem funcional HASKELL onde tais esquemas fossem disponibilizados como combinadores de programas.

O terceiro objectivo acima mencionado conduziu à escolha de duas áreas: a das *estruturas de dados infinitas* e a das *álgebras de processos*. Nelas foram conduzidos os estudos de caso que constituem o cerne da dissertação. O primeiro consiste numa comparação entre definições e provas coindutivas por cálculo e a abordagem clássica por construção de bissimulações, a partir de um estudo sobre sequências infinitas (*streams*) de J. J. M. M. Rutten apresentado em [Rut05]. O segundo prossegue o trabalho iniciado em [Bar01b] em torno da especificação coindutiva de álgebras de processos. Quer num caso, quer noutro, o estudo dos diversos esquemas de programação coindutiva é acompanhado pela sua codificação em HASKELL.

Como seria de esperar numa dissertação de Mestrado, não contém este trabalho resultados propriamente originais. No entanto, parece-nos possível identificar como contribuições mais relevantes, para além da experimentação em coindução por cálculo e da sua animação em HASKELL, a definição coindutiva de operadores de interrupção sobre processos, a transposição relacional de determinados tipos de coalgebras e uma caracterização própria da bissimulação fraca.

Após esta introdução, o capítulo 2 introduz os fundamentos matemáticos deste trabalho, as definições e notação necessária. Os dois estudos de caso referidos ocupam, respectivamente, os capítulos 3 e 4. Por fim, o capítulo 5 conclui e enumera alguns pontos para trabalho futuro. Em apêndice, anexa-se o código de animação desenvolvido em HASKELL.

Capítulo 2

Fundamentos

Resumo

Este capítulo constitui uma breve introdução aos fundamentos teóricos da dissertação, nomeadamente os que se relacionam com a teoria das coalgebras e a coindução. O contexto categorial em que esta última é formulada torna necessária uma revisão rápida de noções elementares de teoria das categorias. Por fim, a exploração da transposição de funções em relações binárias, a que se recorre no capítulo 4 para caracterizar equivalências em álgebra de processos, justifica uma incursão no cálculo relacional. Em qualquer um destes tópicos a abordagem feita é limitada aos conceitos e resultados que serão usados na dissertação. Uma atenção particular é dada às propriedades calculacionais subjacentes.

2.1 Introdução

Os termos *álgebra* e *coalgebra* são usados para descrever uma enorme diversidade de estruturas formais quer em Matemática quer em Ciências da Computação. No capítulo anterior apontou-se já a simetria básica que as relaciona: enquanto as primeiras oferecem descrições abstractas de objectos definidos por *construção*, as segundas revelam-se adequadas para a descrição do comportamento de sistemas a que se tem acesso através de diferentes formas de *observação*. No entanto, esta simetria apenas se torna efectivamente clara quando formulada no nível de abstracção e generalidade proporcionado pela teoria das categorias [Mac71, AHS90]. Nesse contexto a simetria intuitivamente apercebida transforma-se numa dualidade matemática precisa e as noções associadas de *indução* e *coindução* surgem caracterizadas por propriedades universais. Propriedades universais em que, por sua vez, se baseiam as leis base do cálculo indutivo e coindutivo que nos propomos explorar nesta dissertação.

Assim, este capítulo, que tem por objectivo introduzir os conceitos e notação básicos da dissertação, inicia-se pela revisão de noções elementares da teoria das categorias. A secção 2.3 é dedicada à teoria das coalgebras e, por fim, na secção 2.4, introduzem-se os fundamentos do cálculo das relações binárias a que iremos recorrer, no capítulo 4, para estudar um universo de estruturas coalgébricas que é isomorfo ao das relações binárias. Em cada um dos casos o leitor é referido para outros textos para uma exposição detalhada e integrada sobre cada um destes assuntos. Sugestões possíveis são, entre várias outras [Mac71, AHS90, Bor94, BW90] ou [McL92] para a teoria das categorias, [Rut00, JR97] ou [Gum99] para as coalgebras, e [BH93] ou (a segunda parte de) [BM97] para o cálculo das relações.

2.2 Categorias e Propriedades Universais

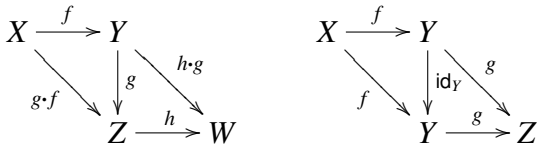
De uma forma muito geral podemos dizer que a teoria das categorias lida com transformações entre estruturas e sua composição, do mesmo modo que a teoria dos conjuntos lida com objectos, a sua agregação em colecções e a relação de pertença. Formalmente,

Definição 1 *Uma categoria C consiste:*

- *numa classe* $obj(C)$ *de objectos* $X, Y, Z, \ldots$
- *para cada par de objectos X e Y, um conjunto* $\mathsf{C}[X,Y]$ *de setas (morfismos) de X para Y*
- *para cada triplo X, Y e Z, uma lei de composição*

$$\cdot : \mathsf{C}[X,Z] \longleftarrow \mathsf{C}[Y,Z] \times \mathsf{C}[X,Y]$$

- *associativa e dotada de elemento identidade*



i.e.,

$$h \cdot g \cdot f = h \cdot (g \cdot f) \tag{2.1}$$

$$\mathsf{id}_Y \cdot f = f \tag{2.2}$$

$$g \cdot \mathsf{id}_Y = g \tag{2.3}$$

- *As setas são determinadas univocamente pelos objectos que ligam,* i.e.*, os conjuntos* $\mathsf{C}[X, Y]$ *são disjuntos dois a dois.*

Conjuntos e funções totais, conjuntos e relações binárias, grupos e seus homomorfismos, espaços topológicos e homeomorfismos, ordens e funções monótonas, termos e provas, são alguns, entre muitos outros exemplos de categorias. Os dois primeiros, a que recorreremos nesta dissertação, são designados por Set e Rel, respectivamente.

Uma vez introduzida uma estrutura matemática, o próximo passo é sempre definir uma noção de transformação que a preserve. Um morfismo entre categorias designa-se por *functor*.

Definição 2 *Um* functor $\mathsf{T} : \mathsf{D} \longleftarrow \mathsf{C}$ *de uma categoria* C *para outra categoria* D *é um par de correspondências que a cada objecto* X *de* C *faz corresponder um objecto* $\mathsf{T}\ X$ *em* D *e a cada* $f \in \mathsf{C}[X, Y]$ *uma seta* $\mathsf{T}\ f \in \mathsf{D}[\mathsf{T}\ X, \mathsf{T}\ Y]$ *tais que*

$$\mathsf{T}\ \mathsf{id}_X = \mathsf{id}_{\mathsf{T}\ X} \tag{2.4}$$

$$\mathsf{T}\ (f \cdot g) = \mathsf{T}\ f \cdot \mathsf{T}\ g \tag{2.5}$$

O exemplo mais elementar de functor é o functor identidade Id. Outro exemplo familiar é o functor $\mathcal{P} : \mathsf{Set} \longleftarrow \mathsf{Set}$ que mapeia um conjunto no seu conjunto potência e cada função $f : Y \longleftarrow X$ na função

$$\mathcal{P}f = \{f\ x |\ x \in X\}$$

Os functores podem ser vistos não só como setas entre categorias, mas também como objectos de outras categorias, providos das definições convenientes de morfismo. Isto é exactamente o que uma *transformação natural* é. Formalmente,

Definição 3 *Dados dois functores* $\mathsf{T}, \mathsf{S} : \mathsf{D} \longleftarrow \mathsf{C}$ *uma* transformação natural $\mathsf{S} : \mathsf{T} \Longleftarrow \mathsf{T}$ *é uma família de* D*-setas, indexadas por objectos de* C*, tais que, para toda* C*-seta* $f : Y \longleftarrow X$ *o seguinte diagrama comuta:*

$$\begin{array}{ccc} X & & \mathsf{T}\ X \xrightarrow{\sigma_X} \mathsf{S}\ X \\ {\scriptstyle f}\downarrow & \qquad & {\scriptstyle \mathsf{T}\ f}\downarrow \qquad\qquad \downarrow{\scriptstyle \mathsf{S}\ f} \\ Y & & \mathsf{T}\ Y \xrightarrow[\sigma_Y]{} \mathsf{S}\ Y \end{array}$$

i.e.,

$$\sigma_Y \cdot \mathsf{T}\ f = \mathsf{S}\ f \cdot \sigma_X \tag{2.6}$$

Cada σ_X *é remetido como o* componente *de* σ *no objecto* X.

Em cálculo de programas a ideia de transformação natural está associada à de polimorfismo. Por exemplo, e retendo-nos apenas a construções elementares em conjuntos, repare-se como tanto a reunião de conjuntos ($\cup$) como a função que forma um conjunto singular (sing) são independentes dos conjuntos ou elementos a que se aplicam — a sua definição é sempre uniforme, o que tecnicamente se exprime por uma propriedade natural. A comutatividade dos diagramas seguintes, que se traduz nas igualdades

$$\cup \cdot (\mathcal{P}f \times \mathcal{P}f) = \mathcal{P}f \cdot \cup \tag{2.7}$$

$$\mathsf{sing} \cdot f = \mathcal{P}f \cdot \mathsf{sing} \tag{2.8}$$

exprime exactamente essas propriedades.

$$\begin{array}{ccc} \mathcal{P}A \times \mathcal{P}A & \xrightarrow{\cup} & \mathcal{P}A \\ {\scriptstyle \mathcal{P}f \times \mathcal{P}f}\downarrow & & \downarrow{\scriptstyle \mathcal{P}f} \\ \mathcal{P}B \times \mathcal{P}B & \xrightarrow{\cup} & \mathcal{P}B \end{array} \qquad \begin{array}{ccc} A & \xrightarrow{\mathsf{sing}} & \mathcal{P}A \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \mathcal{P}f} \\ B & \xrightarrow{\mathsf{sing}} & \mathcal{P}B \end{array}$$

Um outro tipo de transformações naturais a que teremos necessidade de recorrer nesta dissertação são as que provocam a internalização de contexto num dado functor, habitualmente designadas em terminologia anglo-saxónica por *right* (τ_r) e *left* (τ_l) *strength*. A comutatividade dos diagramas seguintes caracterizam-nas para um functor arbitrário T.

$$\begin{array}{ccc} \mathsf{T}X \times C & \xrightarrow{\tau_r} & \mathsf{T}(X \times C) \\ {\scriptstyle \mathsf{T}f \times \mathsf{id}}\downarrow & & \downarrow{\scriptstyle \mathsf{T}(f \times \mathsf{id})} \\ \mathsf{T}Y \times C & \xrightarrow{\tau_r} & \mathsf{T}(Y \times C) \end{array} \qquad \begin{array}{ccc} C \times \mathsf{T}X & \xrightarrow{\tau_l} & \mathsf{T}(C \times X) \\ {\scriptstyle \mathsf{id} \times \mathsf{T}f}\downarrow & & \downarrow{\scriptstyle \mathsf{T}(\mathsf{id} \times f)} \\ C \times \mathsf{T}Y & \xrightarrow{\tau_l} & \mathsf{T}(C \times Y) \end{array}$$

ou seja,

$$\tau_r \cdot (\mathsf{T}f \times \mathsf{id}) = \mathsf{T}(f \times \mathsf{id}) \cdot \tau_r \tag{2.9}$$

$$\tau_l \cdot (f \times \mathsf{Tid}) = \mathsf{T}(f \times \mathsf{id}) \cdot \tau_l \tag{2.10}$$

Não é difícil, porém, concretizar estas transformações para casos específicos. Para o functor $\mathcal{P}$, por exemplo, teremos

$$\tau_r\,(X, c) = \{(x, c) |\ x \in X\} \tag{2.11}$$

O estudo destas transformações naturais é muito vasto e tem diversas implicações na Matemática e, em particular, no cálculo de programas, sobretudo na sua articulação com os monads [Mog91, BW85]. Referências base para esse estudo são [Koc72, Kel82]; [Bar01a] prova um conjunto de regras de cálculo envolvendo τ_r,

τ_l e suas combinações, que se revelam úteis no cálculo de programas e que, num e noutro passo, utilizaremos nesta dissertação.

Um dos tópicos centrais em teoria das categorias é o estudo das propriedades *universais*. Intuitivamente, classificamos uma entidade ϵ como universal no seio de uma família de entidades similares se qualquer outro membro da família a ela puder ser conduzido. Por exemplo, um objecto $\mathbf{1}$ é dito *final* numa categoria se existir um único morfismo para ele a partir de qualquer outro objecto. Quer isto dizer que existe, então, uma forma canónica de relacionar um objecto arbitrário da categoria com $\mathbf{1}$ — a *finalidade* é, pois, uma propriedade universal. A sua dual (*i.e.*, a existência de um objecto a partir do qual existe um único morfismo para qualquer outro), conhecida por *inicialidade*, é ainda uma propriedade universal. De facto, as construções universais surgem sempre aos pares.

Um outro aspecto muito relevante para os propósitos desta dissertação é o facto de a *universalidade*, tal como, de resto, as outras propriedades básicas estudadas em teoria das categorias (a saber, a *functoralidade* e a *naturalidade*), serem formuláveis como *regras de cálculo*. Isto é, como leis, geralmente em formato equacional, para manipular os objectos e setas da categoria.

Introduzimos, de seguida, algumas construções universais que serão úteis nos capítulos seguintes, formulando-as precisamente no estilo calculacional referido.

Definição 4 *O objecto* final *(respectivamente,* inicial*) numa categoria, caso exista, é caracterizado pela existência de um único morfismo* $!_X : \mathbf{1} \longleftarrow X$ *(respectivamente,* $?_X : X \longleftarrow \emptyset$*) a partir de (respectivamente, para) qualquer outro objecto* X*, de forma que para qualquer morfismo* $f : Y \longleftarrow X$ *se verifica* $f : Y \longleftarrow X$*:*

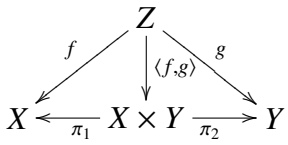
$$!_Y \cdot f = !_X \tag{2.12}$$

$$f \cdot ?_X = ?_Y \tag{2.13}$$

Claramente em Set o objecto inicial é o conjunto vazio $\emptyset$ e o final é (a classe de isomorfismo do) conjunto singular $\mathbf{1}$. As próximas definições generalizam as usuais noções de, respectivamente, produto cartesiano e união disjunta de conjuntos.

Definição 5 *O* produto *entre dois objectos* X *e* Y *na categoria* C *é um objecto* $X \times Y$ *definido com base em* $\pi_1 : X \longleftarrow X \times Y$ *e* $\pi_2 : Y \longleftarrow X \times Y$*, chamadas* projecções*, que satisfazem a propriedade universal: para todo o* $Z \in \text{obj}(\mathsf{C})$ *e as funções* $f : X \longleftarrow Z$ *e* $g : Y \longleftarrow Z$*, existe e é única a função* $\langle f, g \rangle : X \times Y \longleftarrow Z$*,*

usualmente chamada split *de f e g, que faz comutar o diagrama:*



A propriedade universal pode ser escrita como

$$k = \langle f, g\rangle \quad \Leftrightarrow \quad \pi_1 \cdot k = f \ \wedge \ \pi_2 \cdot k = g \tag{2.14}$$

onde $\Rightarrow$ corresponde à *existência* e $\Leftarrow$ corresponde à *unicidade*. De (2.14) facilmente se derivam as seguintes regras de cálculo:

$$\pi_1 \cdot \langle f, g\rangle = f \ , \pi_2 \cdot \langle f, g\rangle = g \tag{2.15}$$

$$\langle \pi_1, \pi_2\rangle \ = \ \mathsf{id}_{X\times Y} \tag{2.16}$$

$$\langle g, h\rangle \cdot f \ = \ \langle g \cdot f, h \cdot f\rangle \tag{2.17}$$

Igualmente, de (2.14) podemos concluir a igualdade de splits:

$$\langle f, g\rangle = \langle h, j\rangle \ \equiv \ f = h \ \wedge \ g = j \tag{2.18}$$

As leis derivadas da propriedade universal são, de acordo com uma convenção que se tem popularizado na comunidade dos cálculos de programas (da qual [BM97] é a referência mais conhecida), identificadas de acordo com o seu efeito na manipulação de expressões. Assim (2.15) é chamada a lei de *cancelamento* (porque cancela o *split*), (2.16) é a lei de *reflexão* e (2.17), que internaliza um composição num *split*, é dita a lei de *fusão*.

A construção produto é igualmente definível sobre os morfismos, transformando $\times$ num functor, via

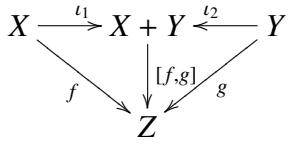
$$f \times g \ \triangleq \ \langle f \cdot \pi_1, g \cdot \pi_2\rangle \tag{2.19}$$

Por último, registe-se a regra de interacção entre o produto de morfismos e a seta universal, regra conhecida como lei de *absorção*:

$$(i \times j) \cdot \langle g, h\rangle \ = \ \langle i \cdot g, j \cdot h\rangle \tag{2.20}$$

Definição 6 *A* soma*, ou o* coproduto*, de X e Y na categoria* C *é a construção dual, que é o mesmo que dizer que, um objecto $X+Y$ pode ser definido por duas funções $\iota_1 : X + Y \longleftarrow X$ e $\iota_2 : X + Y \longleftarrow Y$, designadas de* injecções*, que satisfazem a propriedade universal: para todo $Z \in$ obj(C) e para as funções $f : Z \longleftarrow X$ e*

$g : Z \longleftarrow Y$, *existe um única função* $[f, g] : Z \longleftarrow X + Y$, *usualmente denominada de* either *de* f *e* g, *que comuta o diagrama:*



Assim, podemos reescrever a propriedade universal

$$k = [f, g] \quad \Leftrightarrow \quad k \cdot \iota_1 = f \ \wedge \ k \cdot \iota_2 = g \tag{2.21}$$

que infere *cancelamento*, *reflexão* e *fusão*, respectivamente:

$$[f, g] \cdot \iota_1 = f \ , [f, g] \cdot \iota_2 = g \tag{2.22}$$

$$[\iota_1, \iota_2] \ = \ \mathsf{id}_{X+Y} \tag{2.23}$$

$$f \cdot [g, h] \ = \ [f \cdot g, f \cdot h] \tag{2.24}$$

Novamente a construção estende-se aos morfismos

$$f + g \ \triangleq \ [\iota_1 \cdot f, \iota_2 \cdot g] \tag{2.25}$$

e verifica uma lei de *absorção*

$$[g, h] \cdot (i + j) \ = [g \cdot i, h \cdot j] \tag{2.26}$$

Somas e produtos interagem através da seguinte lei de *troca*

$$[\langle f, g\rangle, \langle f', g'\rangle] \ = \ \langle [f, f'], [g, g']\rangle \tag{2.27}$$

que pode ser provada a partir da propriedade universal quer do produto (2.14) quer da soma (2.21).

A generalização categorial da noção de espaço de funções em Set é também dada por uma propriedade universal. Essa propriedade caracteriza o objecto X^Y (que representa os morfismos de Y para X) por um isomorfismo, designado por *curry*, em homenagem ao matemático Haskell Curry, e representado por um traço horizontal, entre morfismos $f : X \longleftarrow Z \times Y$ e $\overline{f} : X^Y \longleftarrow Z$ de forma que

$$f \ = \ \mathsf{ev} \cdot (\overline{f} \times \mathsf{id}) \tag{2.28}$$

As categorias são classificadas de acordo com a estrutura que exibem. Por exemplo a existência de produtos e objecto final identifica as categorias ditas *cartesianas*. Repare-se que o produto tem, a menos de um isomorfismo, a estrutura de

um monóide abeliano. Assim, as noções correspondentes de associatividade, comutatividade e identidades à esquerda e à direita são captadas por transformações naturais cujas componentes são isomorfismos, e que podem ser definidas com os morfismos elementares já introduzidos. Por exemplo,

$$\begin{aligned} \mathsf{a} : A \times (B \times C) \longleftarrow A \times B \times C &\triangleq \langle \pi_1, \langle \pi_2 \cdot \pi_1, \pi_2 \rangle \rangle \\ \mathsf{s} : B \times A \longleftarrow A \times B &\triangleq \langle \pi_2, \pi_1 \rangle \\ \mathsf{r} : A \longleftarrow \mathbf{1} \times A &\triangleq \langle !_A, \mathsf{id}_A \rangle^\circ \\ \mathsf{l} : A \longleftarrow A \times \mathbf{1} &\triangleq \langle \mathsf{id}_A, !_A \rangle^\circ \end{aligned}$$

A existência de exponenciais torna a categoria cartesiana *fechada*, enquanto a possibilidade de distribuição do produto sobre somas finitas a classifica como *distributiva*. Um dos interesses dessa estrutura distributiva reside na definição de *condicionais*, um tipo de morfismo muito útil em cálculo de programas. Assim, nesta dissertação iremos adoptar o construtor condicional de McCarthy representado por $(p \rightarrow f, g)$, onde $p : \mathbb{B} \longleftarrow A$ é um predicado. Intuitivamente, $(p \rightarrow f, g)$ reduz para f se p avalia em true e para g caso contrário. O construtor é definido como

$$(p \rightarrow f, g) = \langle f, g \rangle \cdot p?$$

onde $p? : A + A \longleftarrow A$ é determinado pelo predicado p como se segue

$$p? = \qquad A \xrightarrow{[\mathsf{id}, p]} A \times (\mathbf{1} + \mathbf{1}) \xrightarrow{\mathsf{dl}} A \times \mathbf{1} + A \times \mathbf{1} \xrightarrow{\pi_1 + \pi_1} A + A$$

Um conjunto amplo de leis para calcular com condicionais é provado em [Gib97]. Em particular, necessitaremos nesta dissertação das seguintes leis de *fusão*:

$$h \cdot (p \rightarrow f, g) = (p \rightarrow h \cdot f, h \cdot g) \tag{2.29}$$

$$(p \rightarrow f, g) \cdot h = (p \cdot h \rightarrow f \cdot h, g \cdot h) \tag{2.30}$$

2.3 Coalgebras e Coindução

Como referimos no capítulo anterior, a diferença essencial entre *álgebras* e *coalgebras* reside no facto de as primeiras representarem sistemas em termos dos seus modos (operações) de *construção*, enquanto as segundas recorrem a métodos de decomposição e *observação*. A definição corrente de álgebra descreve-a como um conjunto equipado com uma colecção de operações. Por exemplo, a álgebra das sequências (finitas) do tipo A, pode ser descrita por um conjunto A^* e dois

construtores

$$\begin{aligned} \mathsf{nil} &: A^* \leftarrow \mathbf{1} \\ \mathsf{cons} &: A^* \leftarrow A \times A^* \end{aligned}$$

Recorrendo à noção de functor para captar a assinatura destes construtores, a mesma álgebra pode ser expressa por uma única função onde o conectivo *either* exprime os dois construtores como modos alternativos de produzir sequências. O par

$$(A^*, [\mathsf{nil}, \mathsf{cons}] \ : A^* \leftarrow \mathbf{1} + A \times A^*) \tag{2.31}$$

constitui uma álgebra para o functor $\mathbf{1} + A \times \mathsf{Id}$, em rigor a álgebra inicial na categoria de todas as álgebras para esse functor.

Dualmente uma coalgebra define-se como

Definição 7 *Uma* coalgebra *para um functor* T *é um par* (C, γ)*, onde* C *é um conjunto (dito o portador da coalgebra ou o espaço de estados) e* $\gamma : \mathsf{T}C \longleftarrow C$ *é uma função. Uma coalgebra será dita* final *se for o objecto final na categoria das coalgebras para esse functor.*

Como exemplo, consideremos a definição do conjunto A^ω de sequências infinitas de um dado tipo A. Ao contrário do caso finito, não há maneira de as construirmos a partir de operações elementares — seria sempre necessário partir já de uma sequência infinita. No entanto, podemos observá-las, distinguindo, em particular o elemento posicionado à cabeça e o resto da sequência. Para isso definimos dois observadores

$$\begin{aligned} \mathsf{hd} &: A \leftarrow A^\omega \\ \mathsf{tl} &: A^\omega \leftarrow A^\omega \end{aligned}$$

que se podem agregar, multiplicativamente, numa coalgebra

$$(A^\omega, \langle \mathsf{hd}, \mathsf{tl} \rangle \ : A \times A^\omega \leftarrow A^\omega) \tag{2.32}$$

Repare-se como a agregação feita por um *split* sublinha a possibilidade de observações diferentes serem possíveis em simultâneo — o que, como é óbvio, não se passa para os construtores. A coalgebra (2.32), que é final na categoria das coalgebras para este functor, será objecto de estudo no próximo capítulo.

Como é evidente para falar de inicialidade ou finalidade torna-se necessário definir o que se entende por morfismo entre coalgebras. Trata-se de uma função entre os conjuntos portadores que preserva a estrutura de observações. Formalmente,

Definição 8 *Um morfismo entre duas* T*-coalgebras,* $\langle U, \alpha \rangle$ *e* $\langle V, \beta \rangle$*, é um morfismo* $h : V \longleftarrow U$ *entre os respectivos portadores tal que o seguinte diagrama comuta*

$$\begin{array}{ccc} U & \xrightarrow{\alpha} & \mathsf{T}\, U \\ {\scriptstyle h}\downarrow & & \downarrow{\scriptstyle \mathsf{T}\, h} \\ V & \xrightarrow{\beta} & \mathsf{T}\, V \end{array}$$

Indução e coindução são os princípios de definição e prova associados, respectivamente, às álgebras iniciais e às coalgebras finais. No resto desta dissertação iremos abordar a definição e prova coindutiva numa perspectiva que explora de forma calculacional a propriedade universal associada. Por agora limitamo-nos a uma breve introdução a estes princípios.

A definição *indutiva* procede por especificação do morfismo que se pretende definir em termos do seu efeito nos construtores da álgebra que, por sua vez, definem as restrições equacionais que determinam univocamente esse morfismo. Atente-se, por exemplo, na definição indutiva da função $\mathsf{length} : \mathbb{N} \longleftarrow A^*$

$$\begin{aligned} \mathsf{length}\ \mathsf{nil} &= 0 \\ \mathsf{length}\ \mathsf{cons}(h, t) &= 1 + \mathsf{length}\ t \end{aligned}$$

Dualmente, a definição por coindução especifica o efeito dos observadores quando aplicados ao morfismo que se pretende definir. Assim, a definição coindutiva da função $\mathsf{merge} : A^\omega \longleftarrow A^\omega \times A^\omega$ é dada por

$$\begin{aligned} \mathsf{hd}\ \mathsf{merge}(s, t) &= \mathsf{hd}\ s \\ \mathsf{tl}\ \mathsf{merge}(s, t) &= \mathsf{merge}(t, \mathsf{tl}\ s) \end{aligned}$$

Enquanto princípio de prova, a indução baseia-se no facto da álgebra inicial ser *mínima*, *i.e.*, não exibir subalgebras diferentes de si própria. Assim, qualquer subalgebra que se defina, por exemplo, a partir da satisfação de um predicado, coincide com a própria álgebra inicial o que estabelece a validade do predicado sobre todos os elementos do portador. Assim, para provar um predicado sobre sequências finitas é suficiente mostrar que ele é fechado para os dois construtores: se assim for, temos a garantia da sua validade abarcar todas as sequências.

Dualmente, o princípio da coindução explora o facto de uma coalgebra final ser *simples*, *i.e.*, não exibir quocientes diferentes de si própria. Isto implica que toda a bissimulação definida sobre o portador da coalgebra final seja um fragmento da relação identidade. Assim, para mostrar que dois elementos de uma coalgebra coincidem, basta exibir uma bissimulação que os relacione. Temos, então, a

garantia de mapearem no mesmo valor na coalgebra final ou, como é usual dizer em Ciências da Computação, no mesmo comportamento.

A noção de *bissimulação* é central em teoria das coalgebras, onde desempenha um papel similar ao das relações compatíveis em álgebra. Intuitivamente, uma bissimulação relaciona dois estados dos portadores de duas coalgebras (distintas ou não) se o resultado da aplicação dos observadores a ambos for idêntico e esse facto for mantido ao longo de todas as possíveis transições das coalgebras. Formalmente,

Definição 9 *Sejam* (C,γ) *e* (D,δ) T*-coalgebras. Uma relação* $R \subseteq C \times D$ *é uma* bissimulação*, se existir* $\beta : \mathsf{T}R \longleftarrow R$ *fazendo* $\pi_1 : (C,\gamma) \longleftarrow (R,\beta)$ *e* $\pi_2 : (D,\gamma) \longleftarrow (R,\beta)$ *morfismos entre coalgebras. Dizemos que* $(c,d) \in C \times D$ *são* bissimilares*, se existir uma bissimulação* $R \subseteq C \times D$ *tal que* $(c,d) \in R$*.*

A noção de bissimulação pode ser expressa através da comutatividade do seguinte diagrama

$$\begin{array}{ccccc} C & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & D \\ \downarrow{\scriptstyle\gamma} & & \downarrow{\scriptstyle\beta} & & \downarrow{\scriptstyle\delta} \\ \mathsf{T}\,C & \xleftarrow{\mathsf{T}\,\pi_1} & \mathsf{T}\,R & \xrightarrow{\mathsf{T}\,\pi_2} & \mathsf{T}\,D \end{array}$$

A relação de bissimilaridade é também chamada *equivalência comportamental* e, usualmente, representada por $\sim$.
Recorde-se que a noção de bissimulação foi introduzida por [Par81] e [Mil80] no contexto das *álgebras de processos*, assunto que constituirá o segundo estudo de caso desta dissertação, numa forma que é um caso muito particular da definição anterior. Em rigor, trata-se da bissimulação para o functor $\mathcal{P}(Act \times \mathsf{Id})$, que trabalharemos no capítulo 4. Mais tarde, em [AM88], foi formulada a definição categorial que acima reproduzimos e que se aplica a qualquer tipo de coalgebras. Por exemplo, podemos concretiza-la para o caso do functor $A \times \mathsf{Id}$ que capta a assinatura dos observadores de sequências infinitas. Este functor será objecto do primeiro estudo de caso desta dissertação, no capítulo 3.

Definição 10 *Uma bissimulação para o functor* $A \times \mathsf{Id}$ *sobre o portador* A^ω *é uma relação* $R \subseteq A^\omega \times A^\omega$ *tal que, para todo* σ *e* τ *em* A^ω*, se* $\langle \sigma, \tau \rangle \in R$
então

1. $\mathsf{hd}\,\sigma = \mathsf{hd}\,\tau$
2. $(\mathsf{tl}\,\sigma, \mathsf{tl}\,\tau) \in R$

2.4 Cálculo de Relações Binárias

O cálculo relacional, que explora a estrutura da categoria Rel dos conjuntos e relações binárias, tem vindo a ganhar uma importância crescente no cálculo de programas. Em primeiro lugar porque, ao contrário das funções, as relações são essencialmente não determinísticas e são, portanto, candidatas naturais à modelação de problemas também eles não determinísticos associados aos fenómonos computacionais. Depois porque o facto de cada relação ter uma relação conversa bem definida permite especificar problemas em termos de conversos de outros problemas. Finalmente porque o poder expressivo do cálculo e a riqueza da sua estrutura formal, expressa numa enorme quantidade de leis, conduz a provas mais concisas e elegantes.

Nesta dissertação faremos um uso limitado do cálculo relacional — basicamente apenas nas últimas secções do capítulo 4, que, contudo, ilustram bem o seu poder expressivo. Justifica-se, por isso, uma breve resenha deste assunto. O leitor interessado é remetido para [BM97] e [BH93] para um tratamento mais amplo e aprofundado.

Representamos por $R : B \longleftarrow A$ uma relação binária de A para B, e por bRa o facto $\langle b, a\rangle \in R$. O conjunto de todas as relações de A para B está ordenado pela inclusão $\subseteq$, sendo a igualdade estabelecida por anti-simetria. O facto $R \subseteq S$ significa que a relação S é mais definida ou menos determinística que R. Dito de outra forma, que para todo a e b dos tipos apropriados, $bRa \Rightarrow bS\,a$.

A álgebra das relações é construída sobre três operadores básicos: a composição relacional ($R \cdot S$), a intersecção ($R \cap S$) e o converso (R°). Como se esperaria, $aR^\circ b$ sse bRa, a intersecção corresponde à intersecção nos conjuntos, e $\cdot$ generaliza a composição de funções: $b(R \cdot S)c$ sse existir um e $a \in A$ tal que $bRa \wedge aS\,c$.

Toda a função f pode ser vista como a relação dada pelo seu grafo e que, nesta dissertação será igualmente denotada por f. Assim, $bfa \;\equiv\; b = fa$. Conversamente, qualquer relação $R : B \longleftarrow A$ pode ser transposta de forma única para uma função $\Lambda R : \mathcal{P}B \longleftarrow A$, onde o operador de transposição Λ satisfaz a seguinte propriedade universal:

$$f = \Lambda R \;\equiv\; (bRa \;\equiv\; b \in (fa)) \tag{2.33}$$

A última parte do capítulo 4 baseia-se, essencialmente, neste isomorfismo que satisfaz, entre outras, as seguintes leis de *reflexão*

$$\Lambda \in = \mathsf{id} \tag{2.34}$$

e *fusão*

$$\Lambda\,(f \cdot R) \;=\; \mathcal{P}f \cdot \Lambda\,R \tag{2.35}$$

$$\Lambda\,(R \cdot f) \;=\; \Lambda\,R \cdot f \tag{2.36}$$

A interacção entre funções e relações é origem de um conjunto de propriedades muito importantes na agilização das provas e, em particular, na eliminação de variáveis das expressões. Registe-se, em particular, a regra

$$b\,(f^{\circ} \cdot R \cdot g)\,a \;\equiv\; (f\,b)\,R\,(g\,a) \tag{2.37}$$

O cálculo relacional é constituido por um vasto conjunto de leis que exploram a estrutura categorial (identidade e composição), reticular (inclusão, intersecção, topo ($\top$) e base ($\bot$)) e a sua interacção. Uma maneira económica de classificar e agregar essas leis é oferecida pela noção de conexão de Galois. De uma forma geral, uma conexão de Galois é uma equivalência envolvendo duas ordens, $\leq$ e $\sqsubseteq$, na forma

$$\underbrace{f}_{\text{adjunto inferior}}\; b \leq a \;\equiv\; b \sqsubseteq \underbrace{g}_{\text{adjunto superior}}\; a$$

isto é,

$$f^{\circ} \cdot \leq \;\; = \;\; \sqsubseteq \cdot g$$

que se representa por $f \dashv g$.

O par de adjuntos satisfaz as propriedades de definição mútua, preservação e monotonia expressas no diagrama seguinte

$(f\ b) \leq a \equiv b \sqsubseteq (g\ a)$		
Descrição	f	g
Definição	$f\ b = \bigwedge\{a \mid b \sqsubseteq g\ a\}$	$g\ a = \bigsqcup\{b \mid f\ b \leq a\}$
Cancelamento	$f(g\ a) \leq a$	$b \sqsubseteq g(f\ a)$
Preservação	$f(b \sqcup b') = (f\ b) \vee (f\ b')$	$g(a' \vee a) = (g\ a') \sqcap (g\ a)$
	$f \bot = \bot$	$g \top = \top$
Monotonia	$b \sqsubseteq b' \Rightarrow f\ b \leq f\ b'$	$a \leq a' \Rightarrow g\ a \sqsubseteq g\ a'$

das quais se deriva, em cada caso, o núcleo de leis aplicáveis aos operadores envolvidos. Por exemplo, a conexão

$$X^{\circ} \subseteq Y \;\equiv\; X \subseteq Y^{\circ} \tag{2.38}$$

caracteriza completamente o operador converso, fornecendo como corolários as leis

$$\begin{array}{rl} \textit{Cancelamento} & (R^\circ)^\circ = R \\ \textit{Monotonia} & R \subseteq S \equiv R^\circ \subseteq S^\circ \\ \textit{Distribuição} & (R \cap S)^\circ = R^\circ \cap S^\circ, (R \cup S)^\circ = R^\circ \cup S^\circ \end{array}$$

Outras conexões a que recorreremos nesta dissertação incluem as regras de *shunting*

$$f \cdot R \subseteq S \equiv R \subseteq f^\circ \cdot S \tag{2.39}$$
$$R \cdot f^\circ \subseteq S \equiv R \subseteq S \cdot f \tag{2.40}$$

que correspondem às conexões $(f\cdot) \dashv (f^\circ\cdot)$ e $(\cdot f^\circ) \dashv (\cdot f)$, respectivamente, e ainda as que caracterizam os dois operadores de divisão relacional, $(\cdot R) \dashv (/R)$ e $(R\cdot) \dashv (R\backslash)$, *i.e.*,

$$R \cdot X \subseteq S \equiv X \subseteq R \setminus S \tag{2.41}$$
$$X \cdot R \subseteq S \equiv X \subseteq S \,/\, R \tag{2.42}$$

Recorde-se que estes operadores são classicamente definidos em lógica de 1ª ordem por

$$\begin{array}{rcl} a(R \setminus S)c & \iff & \forall_b \,.\, (bRa) \Rightarrow (bS\,c) \\ c(S \,/\, R)a & \iff & \forall_b \,.\, (aRb) \Rightarrow (cS\,b) \end{array}$$

Capítulo 3

Estudo de Caso: Sequências Infinitas

Resumo

Este capítulo apresenta e compara, através de diversos exemplos de manipulação de streams, dois estilos de definição e prova coindutiva: a construção explícita de bissimulações e o cálculo baseado na propriedade universal. Introduz-se a noção de anamorfismo *(que corresponde à programação por* coiteração*), o apomorfismo (que corresponde à* correcursão*) e o* futumorfismo*. A partir do estudo base realizado atrás, neste capítulo desenvolveu-se, ainda, uma pequena biblioteca em* HASKELL *onde se codificam, na forma de combinadores de programas funcionais, os principais esquemas de definição coindutiva.*

3.1 Definição e Prova por Coindução

A recursão é um conceito central em computação. O objecto da programação coindutiva são os programas *correcursivos*, *i.e.*, aqueles que podem ser especificados por funções cujo codomínio é um tipo *coindutivo*. Os seus duais, a que chamaremos programas propriamente *recursivos*, são porventura de manipulação mais simples e, de longe, mais estudados e conhecidos. Definem-se, como se esperaria, como aqueles que são definidos por funções cujo domínio é um tipo *indutivo*. Assim, a dualidade *recursão/correcursão* é outra face da dualidade *indução/coindução* que, por sua vez, como discutimos na capítulo 2, tem origem na dualidade mais vasta que relaciona *álgebras* e *coalgebras*.

Este primeiro estudo de caso em programação coindutiva tem por objecto um tipo particularmente simples de estrutura infinita, a *stream*, mas que se pode considerar

omnipresente nos sistemas computacionais. Streams modelam o fluxo de dados entre aplicações, com um espectro tão largo que vai desde as componentes de um sistema operativo, as interfaces com o utilizador ou as comunicações entre *web-services*. Além disso são duais das sequências finitas, outra estrutura de dados muito comum em programação, mas que, precisamente por serem finitas, são de natureza *indutiva*, conforme discutido no capítulo anterior.

Como vimos, programas sobre A^* são representados por funções definidas pelo seu efeito nos construtores. As provas recorrem a um princípio de indução que explora essa mesma estrutura de construção.

Exemplo. Recordemos a definição de função length do capítulo 2 e consideremos, ainda, a seguinte função que corresponde à iteração de uma função sobre todos os elementos de uma sequência[1]:

$$\begin{aligned} \mathsf{map}\ f\ [] &= 0 \\ \mathsf{map}\ f(h : t) &= f(h) : \mathsf{map}\ f\ t \end{aligned}$$

A prova indutiva do facto

$$\mathsf{length}(\mathsf{map}\ f\ l) = \mathsf{length}\ l$$

cujo caso de base (para $l = \mathbf{0}$) é trivial, tem o seguinte o passo indutivo

$$\begin{aligned} & \mathsf{length}(\mathsf{map}\ f(h : t)) \\ = & \quad \{ \text{ definição de map } f \} \\ & \mathsf{length}(f(h) : \mathsf{map}\ f\ t) \\ = & \quad \{ \text{ definição de length} \} \\ & 1 + \mathsf{length}(\mathsf{map}\ f\ t) \\ = & \quad \{ \text{ hipótese de indução} \} \\ & 1 + \mathsf{length}\ t \\ = & \quad \{ \text{ definição de length} \} \\ & \mathsf{length}(h : t) \end{aligned}$$

✓

[1] Na tradição da programação funcional [Bir98] tais funções, que correspondem à extensão functorial de um construtor de tipos sobre funções, são designadas por map.

Como se vê neste exemplo, o raciocínio indutivo requer que, através de um processo sistemático de desdobramento da definição, os argumentos se tornem progressivamente mais pequenos, *i.e.*, cada vez mais próximos dos construtores do tipo. Assim, a questão que se coloca quando pretendemos programar com streams e provar propriedades sobres esses programas, é como proceder quando o processo de desdobramento da definição *não* termina.

Existem várias respostas a essa questão (ver [GH05] para uma apresentação sistemática). Nesta dissertação não nos dedicaremos às respostas mais clássicas que exploram a denotação dos programas em determinados domínios semânticos (tipicamente estruturas ordenadas ou topológicas), nomeadamente a *indução sobre ponto fixo* [dB80]. O nosso foco será a *coindução*, uma técnica de prova que, tal como o método indutivo referido acima, explora a semântica *operacional* dos programas.

A observação básica deste método é de que, se é verdade que o processo de desdobramento da definição de uma função correcursiva não termina, ele vai revelando componentes cada vez maiores do resultado. Deste modo, todo o elemento do tipo coindutivo acaba por ficar unicamente determinado ao longo desse processo. Assim, para raciocinar em programação coindutiva a nossa atenção transfere-se da análise estrutural do resultado, típica do raciocínio indutivo, para a progressiva construção do resultado.

Este tipo de provas é, por isto mesmo, baseado na noção de *bissimulação* introduzida no capítulo 2. Intuitivamente, a ideia básica é de que, para provar uma igualdade, se comparam as observações imediatas e se prova que essa relação se mantém à medida que as definições se desdobram.

Como referimos na Introdução a esta dissertação, existem dois estilos de definição coindutiva, que poderíamos caracterizar como *coindução explícita* e *coindução implícita*, respectivamente.

- No primeiro estilo a função é definida explicitamente por uma expressão recursiva que exprime o resultado da aplicação dos observadores do tipo ao seu resultado pela observação dos seus efeitos (ver, como exemplo, a definição da função merge no capítulo 2). O princípio de prova associado é a construção de uma bissimulação que contenha o par formado pelos dois membros da igualdade que se pretende verificar. Como vimos, se tal par existir, isso significa que os dois membros mapeiam no mesmo elemento da coalgebra final e representam, portanto, o mesmo valor coindutivo (ou, mais operacionalmente, exibem o mesmo comportamento). Este é o estilo mais

usado na literatura sobre coalgebras. Por vezes, como veremos, é adoptado um estilo equacional de prova, em que a bissimulação é codificada numa espécie de *hipótese de coindução*, o que resulta em provas elegantes, embora menos genéricas que as que adoptam o estilo alternativo que referimos no próximo item.

- O segundo estilo explora o facto de um tipo coindutivo ser uma coalgebra final e, portanto, exibir uma propriedade universal. Essa propriedade determina um conjunto de regras de cálculo (similares, por exemplo, às que no capítulo anterior associamos à caracterização universal do produto cartesiano) que são usadas para estabelecer a igualdade por raciocínio equacional e *pointfree*, evitando a construção explícita de bissimulações. As definições, por seu lado, são feitas especificando o *gene* da função — *i.e.*, uma coalgebra que explica um passo da sua dinâmica. Este estilo tem origem no cálculo de programas, onde corresponde por dualidade a um estilo de cálculo de programas recursivos por vezes referido como o *formalismo de Bird-Meertens* [Gib02, BM97].

No presente capítulo iremos explorar este segundo estilo na definição de diversas funções sobre streams e na prova de algumas propriedades. Como contrapartida apresentaremos as mesmas definições e provas no estilo de *coindução explícita*. Para este estilo basear-nos-emos na teoria das streams proposta por J.J.M.M. Rutten em [Rut05]. Esse artigo recorre a uma notação que explora o facto de as streams de A, para qualquer A, serem isomorfas ao conjunto das funções de $\mathbb{N}$ para A, *i.e.*,

$$A^\omega \;=\; \{\sigma |\; A \leftarrow \mathbb{N}\}$$

Assim, a cabeça, $\mathsf{hd}(\sigma)$, de uma qualquer stream σ, é definida como o *valor inicial de* σ, representado por $\sigma(0)$. Similarmente, a cauda, $\mathsf{tl}(\sigma)$, é defininda como a *derivada* σ' de σ. Para todo o valor $n \geq 0$ tem-se que,

$$\sigma'(n) \;=\; \sigma(n+1)$$

i.e.,

$$\mathsf{tl}(\sigma) \;=\; \sigma' = (\sigma(1), \sigma(2), \sigma(3), \ldots)$$

O acesso aos elementos de σ é dado pela seguinte notação:

$$\sigma \;=\; (\sigma(0), \sigma(1), \sigma(2), \ldots)$$

Por fim, a definição de bissimulação apresentada na definição 10 é re-escrita de forma equivalente, como

Definição 11 *Uma bissimulação em A^ω é uma relação $R \subseteq A^\omega \times A^\omega$ tal que, para todo σ e τ em A^ω, se $\langle \sigma, \tau \rangle \in R$*

1. $\sigma\ (0) = \tau\ (0)$
2. $\langle \sigma', \tau' \rangle \in R$

Nas próximas secções estudaremos os princípios de definição e prova por cálculo associados à coindução na sua forma mais simples, também dita *estrutural*, a que corresponde a propriedade universal que caracteriza o *anamorfismo* [BM97]. No entanto, este tipo de coindução não é suficientemente genérico para abarcar todos os tipos de coindução. Assim, de seguida, apresentaremos dois outros esquemas coindutivos igualmente caracterizados por propriedades universais: o *apomorfismo*, introduzido por [VU97], e o *futumorfismo*, introduzido por [UV99]. Todos estes esquemas foram incorporados numa pequena biblioteca em HASKELL a que recorremos para os ilustrar.

3.2 Anamorfismo

O estilo *coindução implícita* ou *por cálculo* é baseado na propriedade universal que caracteriza, para um dado functor T, a T-coalgebra final. Formalmente,

Definição 12 *Seja ν_{T} o portador da coalgebra final $\mathsf{out}_{\mathsf{T}}$. Um* anamorfismo *é o único morfismo entre T – coalgebras que faz comutar o diagrama:*

$$\begin{array}{ccc} \nu_{\mathsf{T}} & \xrightarrow{\mathsf{out}_{\mathsf{T}}} & \mathsf{T}\ \nu_{\mathsf{T}} \\ {\scriptstyle [\![(p)]\!]_{\mathsf{T}}}\uparrow & & \uparrow{\scriptstyle \mathsf{T}\ [\![(p)]\!]_{\mathsf{T}}} \\ U & \xrightarrow{p} & \mathsf{T}\ U \end{array}$$

A propriedade universal é, equivalentemente, captada pela lei

$$k = [\![(p)]\!] \quad \Leftrightarrow \quad \mathsf{out}_{\mathsf{T}} \cdot k = \mathsf{T}\ k \cdot p \tag{3.1}$$

A lei (3.1) transporta

- um princípio de *definição*, correspondente à implicação da esquerda para a direita;
- um princípio de *prova*, correspondente à implicação inversa.

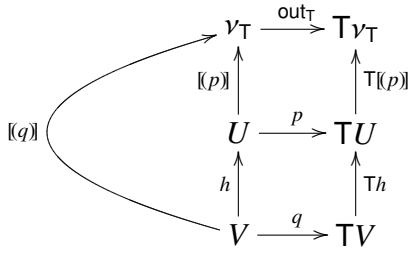
No resto desta secção iremos explorar esses princípios comparando-os com o estilo de definição e prova por coindução explícita. Antes, porém, registemos um conjunto de leis úteis para o cálculo e que correspondem, respectivamente, ao *cancelamento*, *reflexão* e *fusão* associadas ao anamorfismo.

$$\mathsf{out}_{\mathsf{T}} \cdot [\![(p)]\!] \ = \ \mathsf{T}\ [\![(p)]\!] \cdot p \tag{3.2}$$

$$[\![(\mathsf{out}_{\mathsf{T}})]\!] \ = \ \mathsf{id}_{\nu_{\mathsf{T}}} \tag{3.3}$$

$$[\![(p)]\!] \cdot h \ = \ [\![(q)]\!] \quad \text{se} \quad p \cdot h \ = \ \mathsf{T}\ h \cdot q \tag{3.4}$$

Todas elas são facilmente deduzidas de (3.1). Por exemplo, a lei da fusão que diz que um anamorfismo composto com outro morfismo entre coalgebras é ainda uma anamorfismo, como se ilustra no diagrama



prova-se do seguinte modo

Prova.

$$
\begin{array}{ll}
 & [\![p]\!] \cdot h \;=\; [\![q]\!] \\
\equiv & \quad \{ \text{ propriedade universal } \} \\
 & \omega \cdot [\![p]\!] \cdot h \;=\; \mathsf{T}([\![p]\!] \cdot h) \cdot q \\
\equiv & \quad \{ \text{ cancelamento, T functor } \} \\
 & \mathsf{T}[\![p]\!] \cdot p \cdot h \;=\; \mathsf{T}[\![p]\!] \cdot \mathsf{T}h \cdot q \\
\Leftarrow & \quad \{ \text{ igualdade de funções } \} \\
 & p \cdot h \;=\; \mathsf{T}h \cdot q
\end{array}
$$

□

Repare-se que estas definições e provas são absolutamente genéricas, *i.e.*, válidas para qualquer functor T para o qual exista coalgebra final. Voltando ao caso das streams, temos a seguinte definição para a coalgebra final:

$$\omega_{\mathsf{T}} = \langle \mathsf{hd}, \mathsf{tl} \rangle \tag{3.5}$$

3.2.1 Codificação em HASKELL

Para codificarmos este esquema em HASKELL precisamos de definir um construtor, pois um functor em HASKELL é um construtor. Definimos, então, o functor $\mathsf{S} = A \times S$ que capta a assinatura de observadores das streams, usando a classe `Functor` do `Haskell Prelude`.

```
class Functor f where fmap :: (a -> b) -> f a -> f b

instance Functor (S a)
    where fmap f (St c x) = St c (f x)
```

De seguida, o portador da coalgebra final para este functor, *i.e.*, o conjunto A^{ω}, é especificado como o maior ponto fixo. Assim,

```
newtype Mu f = In (f (Mu f))

unIn :: Mu f -> f (Mu f)
unIn (In x) = x

newtype Nu f = Fin (f (Nu f))

unFin :: Nu f -> f (Nu f)
unFin (Fin x) = x
```

Para codificar o anamorfismo, usamos as leis de fusão e cancelamento, definindo-se:

```
ana :: Functor f => (c -> f c) -> c -> Nu f
ana phi = Fin . fmap (ana phi) . phi
```

3.2.2 Definição Coindutiva

Consideremos, agora, a definição de algumas funções sobre streams nos dois estilos de coindução. Os exemplos são tomados de [Rut05].

1. $\boxed{\mathsf{even} : A^{\omega} \longleftarrow A^{\omega}}$
   A stream resultante da aplicação desta função é a stream dos valores das posições pares [2].

[2] É de notar que se aplicarmos esta função à stream dos naturais, ao contrário do que se poderia esperar, vamos obter a stream dos números ímpares, uma vez que se considerou a cabeça como posição 0 e não 1.

Segundo [Rut05] a função define-se por:

$$\begin{aligned} \mathsf{even}(\sigma)(0) &= \sigma(0) \\ \mathsf{even}(\sigma)' &= \mathsf{even}(\sigma'') \end{aligned}$$

A definição pode ser validada mostrando que

$$\mathsf{even}([ab]) = [a]$$

onde a notação $[s]$, para $s \in A^*$, representa a stream obtida por repetição infinita do padrão s. Assim

$$\begin{aligned} \mathsf{even}([ab]) &= \mathsf{even}([ab])(0) : \mathsf{even}([ab])' \\ &= [ab](0) : \mathsf{even}([ab]'') \\ &= a : \mathsf{even}([ab]) \\ &= [a] \end{aligned}$$

Vejamos agora a definição pela propriedade universal. A ideia aqui é definir não a função *toda*, mas o seu *gene*, *i.e.*, a coalgebra que especifica a sua dinâmica num passo de execução e que, por isso, transporta a sua herança genética. Para este exemplo, essa coalgebra é

$$\langle \mathsf{hd}, \mathsf{tl} \cdot \mathsf{tl} \rangle : A \times A^\omega \longleftarrow A^\omega \tag{3.6}$$

Para o verificar, comecemos por representar o diagrama correspondente com even como o seu anamorfismo:

$$\begin{array}{ccc} \mathsf{A}^\omega & \xrightarrow{\quad \mathsf{even} \quad} & \mathsf{A}^\omega \\ {\scriptstyle \langle \mathsf{hd},\mathsf{tl}\cdot\mathsf{tl} \rangle}\downarrow & & \downarrow{\scriptstyle \langle \mathsf{hd},\mathsf{tl} \rangle} \\ \mathsf{A} \times \mathsf{A}^\omega & \xrightarrow{\quad \mathsf{id}\times\mathsf{even} \quad} & \mathsf{A} \times \mathsf{A}^\omega \end{array}$$

Como o diagrama comuta, vem que

$$\begin{array}{ll} & \langle \mathsf{hd}, \mathsf{tl} \rangle \cdot \mathsf{even} \;=\; (\mathsf{id} \times \mathsf{even}) \cdot \langle \mathsf{hd}, \mathsf{tl} \cdot \mathsf{tl} \rangle \\ \equiv & \quad \{ \times\text{-fusão e absorção} \} \\ & \langle \mathsf{hd} \cdot \mathsf{even}, \mathsf{tl} \cdot \mathsf{even} \rangle \;=\; \langle \mathsf{id} \cdot \mathsf{hd}, \mathsf{even} \cdot \mathsf{tl} \cdot \mathsf{tl} \rangle \\ \equiv & \quad \{ \text{definição de identidade} \} \\ & \langle \mathsf{hd} \cdot \mathsf{even}, \mathsf{tl} \cdot \mathsf{even} \rangle \;=\; \langle \mathsf{hd}, \mathsf{even} \cdot \mathsf{tl} \cdot \mathsf{tl} \rangle \\ \equiv & \quad \{ \text{definição de split} \} \\ & \mathsf{hd} \cdot \mathsf{even} \;=\; \mathsf{hd} \\ & \mathsf{tl} \cdot \mathsf{even} \;=\; \mathsf{even} \cdot \mathsf{tl} \cdot \mathsf{tl} \end{array}$$

o que coincide com a definição explícita dada acima. Provamos, portanto, que

$$\mathsf{even} \;=\; [\!(\langle \mathsf{hd}, \mathsf{tl} \cdot \mathsf{tl} \rangle)\!]$$

Vejamos, por fim, como esta definição implícita é directamente codificada em HASKELL, usando o funcional `ana` definido acima:

```
even :: Stream a -> Stream a
even = ana phi
       where phi a = St (headS a) ((tailS . tailS) a)
```

2. $\boxed{\mathsf{odd} : A^\omega \longleftarrow A^\omega}$
Esta função retorna a stream das posições ímpares.

Em [Rut05] define-se

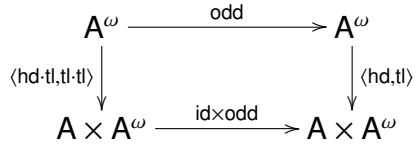
$$\begin{aligned} \mathsf{odd}(\sigma)(0) &= \sigma(1) \\ \mathsf{odd}(\sigma)' &= \mathsf{odd}(\sigma'') \end{aligned}$$

o que pode ser validado provando a seguinte igualdade:

$$\mathsf{odd}([ab]) = [b].$$

$$\begin{aligned} \mathsf{odd}([ab]) &= \mathsf{odd}([ab])(0) : \mathsf{odd}([ab])' \\ &= [ab](1) : \mathsf{odd}([ab]'') \\ &= b : \mathsf{odd}([ab]) \\ &= [b] \end{aligned}$$

Definição pela Propriedade Universal
Comecemos por representar o diagrama:



Como o diagrama comuta, vem que

$$
\begin{aligned}
& \langle \mathsf{hd}, \mathsf{tl} \rangle \cdot \mathsf{odd} \;=\; (\mathsf{id} \times \mathsf{odd}) \cdot \langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \cdot \mathsf{tl} \rangle \\
\equiv & \quad \{ \text{×-fusão e absorção} \} \\
& \langle \mathsf{hd} \cdot \mathsf{odd}, \mathsf{tl} \cdot \mathsf{odd} \rangle \;=\; \langle \mathsf{id} \cdot \mathsf{hd} \cdot \mathsf{tl}, \mathsf{odd} \cdot \mathsf{tl} \cdot \mathsf{tl} \rangle \\
\equiv & \quad \{ \text{definição de identidade} \} \\
& \langle \mathsf{hd} \cdot \mathsf{odd}, \mathsf{tl} \cdot \mathsf{odd} \rangle \;=\; \langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{odd} \cdot \mathsf{tl} \cdot \mathsf{tl} \rangle \\
\equiv & \quad \{ \text{definição de split} \} \\
& \mathsf{hd} \cdot \mathsf{odd} \;=\; \mathsf{hd} \cdot \mathsf{tl} \\
& \mathsf{tl} \cdot \mathsf{odd} \;=\; \mathsf{odd} \cdot \mathsf{tl} \cdot \mathsf{tl}
\end{aligned}
$$

Portanto, $\mathsf{odd} = [\![\langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \cdot \mathsf{tl} \rangle]\!]$.

3. $\boxed{\mathsf{zipS} : (A \times B)^\omega \longleftarrow A^\omega \times B^\omega}$
A função zipS junta dois a dois os elementos de duas streams, fazendo algo semelhante à concatenação, mas que permite sempre saber de que stream veio determinado elemento, bastanto para isso, verificar se se encontra na primeira ou na segunda posição do par.

Segundo Rutten, na referência citada, pode-se dizer que:

$$
\begin{aligned}
\mathsf{zipS}(\sigma, \tau)(0) &= (\sigma(0), \tau(0)) \\
\mathsf{zipS}(\sigma, \tau)' &= \mathsf{zipS}(\sigma', \tau')
\end{aligned}
$$

O que se valida provando a seguinte igualdade:

$$\mathsf{zipS}([a], [b]) = [(a, b)].$$

$$
\begin{aligned}
\mathsf{zipS}([a], [b]) &= ([a](0), [b](0)) : \mathsf{zipS}([a]', [b]') \\
&= (a, b) : \mathsf{zipS}([a], [b]) \\
&= [(a, b)]
\end{aligned}
$$

Definição pela Propriedade Universal
Comecemos por representar o diagrama:

$$\begin{array}{ccc} \mathsf{A}^\omega \times \mathsf{B}^\omega & \xrightarrow{\quad\mathsf{zipS}\quad} & (\mathsf{A}\times\mathsf{B})^\omega \\ \Big\downarrow{\scriptstyle \langle \mathsf{hd}\cdot\pi_1\times\mathsf{hd}\cdot\pi_2,\mathsf{tl}\cdot\pi_1\times\mathsf{tl}\cdot\pi_2\rangle} & & \Big\downarrow{\scriptstyle \langle \mathsf{hd},\mathsf{tl}\rangle} \\ (\mathsf{A}\times\mathsf{B})\times(\mathsf{A}\times\mathsf{B})^\omega & \xrightarrow{\quad\mathsf{id}\times\mathsf{zipS}\quad} & (\mathsf{A}\times\mathsf{B})\times(\mathsf{A}\times\mathsf{B})^\omega \end{array}$$

Como o diagrama comuta, vem que

$$\begin{array}{ll} & \langle \mathsf{hd},\mathsf{tl}\rangle \cdot \mathsf{zipS} \;=\; (\mathsf{id}\times\mathsf{zipS})\cdot\langle \mathsf{hd}\cdot\pi_1\times\mathsf{hd}\cdot\pi_2, \mathsf{tl}\cdot\pi_1\times\mathsf{tl}\cdot\pi_2\rangle \\ \equiv & \quad \{\times\text{-fusão e absorção}\} \\ & \langle \mathsf{hd}\cdot\mathsf{zipS}, \mathsf{tl}\cdot\mathsf{zipS}\rangle \;=\; \langle \mathsf{id}\cdot(\mathsf{hd}\cdot\pi_1\times\mathsf{hd}\cdot\pi_2), \mathsf{zipS}\cdot(\mathsf{tl}\cdot\pi_1\times\mathsf{tl}\cdot\pi_2)\rangle \\ \equiv & \quad \{\text{definição de identidade}\} \\ & \langle \mathsf{hd}\cdot\mathsf{zipS}, \mathsf{tl}\cdot\mathsf{zipS}\rangle \;=\; \langle \mathsf{hd}\cdot\pi_1\times\mathsf{hd}\cdot\pi_2, \mathsf{zipS}\cdot(\mathsf{tl}\cdot\pi_1\times\mathsf{tl}\cdot\pi_2)\rangle \\ \equiv & \quad \{\text{definição de split}\} \\ & \mathsf{hd}\cdot\mathsf{zipS} \;=\; \mathsf{hd}\cdot\pi_1\times\mathsf{hd}\cdot\pi_2 \\ & \mathsf{tl}\cdot\mathsf{zipS} \;=\; \mathsf{zipS}\cdot(\mathsf{tl}\cdot\pi_1\times\mathsf{tl}\cdot\pi_2) \end{array}$$

Portanto, $\mathsf{zipS} = [\![(\langle \mathsf{hd}\cdot\pi_1\times\mathsf{hd}\cdot\pi_2, \mathsf{tl}\cdot\pi_1\times\mathsf{tl}\cdot\pi_2\rangle)]\!]$.

4. $\boxed{\mathsf{concatS} : A^\omega \longleftarrow A^\omega \times A^\omega}$
A função concatS é muito semelhante à função zipS. Nesta, consideram-se as duas streams, juntando-as numa só, em que cada elemento é um par ordenado. Ou seja, como o próprio nome indica, a função zipS funciona como um fecho que junta duas streams, mas possibilitando-nos saber que elementos provêm de uma e de outra stream, respectivamente. Na função concatS, juntam-se as duas streams, alternando os elementos de uma e de outra.

Segundo [Rut05] tem-se:

$$\begin{aligned} \mathsf{concatS}(\sigma,\tau)(0) &= \sigma(0) \\ \mathsf{concatS}(\sigma,\tau)' &= \mathsf{concatS}(\tau,\sigma') \end{aligned}$$

O que se valida provando:

$$\mathsf{concatS}(\sigma,\tau) = \sigma(0) : \mathsf{concatS}(\tau,\sigma').$$

$$\begin{aligned}
\mathsf{concatS}(\sigma,\tau) &= \mathsf{concatS}(\sigma,\tau)(0) : \mathsf{concatS}(\sigma,\tau)' \\
&= \sigma(0) : \mathsf{concatS}(\tau,\sigma') \\
&= \sigma(0) : \mathsf{concatS}(\tau,\sigma')(0) : \mathsf{concatS}(\tau,\sigma')' \\
&= \sigma(0) : \tau(0) : \mathsf{concatS}(\sigma',\tau') \\
&= \sigma(0) : \tau(0) : \mathsf{concatS}(\sigma',\tau')(0) : \mathsf{concatS}(\sigma',\tau')' \\
&= \sigma(0) : \tau(0) : \sigma(1) : \mathsf{concatS}(\tau',\sigma'')
\end{aligned}$$

Deste modo observa-se que:

$$\mathsf{concatS}(\sigma,\tau) \;=\; (\sigma(0),\tau(0),\sigma(1),\tau(1),\sigma(2),\tau(2),\ldots)$$

Definição pela Propriedade Universal
Comecemos por representar o diagrama:

$$\begin{array}{ccc}
\mathsf{A}^\omega \times \mathsf{A}^\omega & \xrightarrow{\quad\mathsf{concatS}\quad} & \mathsf{A}^\omega \\
{\scriptstyle \langle \mathsf{hd}\cdot\pi_1,\mathsf{s}\cdot(\mathsf{tl}\times\mathsf{id})\rangle}\downarrow & & \downarrow{\scriptstyle \langle \mathsf{hd},\mathsf{tl}\rangle} \\
\mathsf{A}\times\mathsf{A}^\omega\times\mathsf{A}^\omega & \xrightarrow{\quad\mathsf{id}\times\mathsf{concatS}\quad} & \mathsf{A}\times\mathsf{A}^\omega
\end{array}$$

Como o diagrama comuta, vem que

$$\begin{aligned}
& \langle \mathsf{hd},\mathsf{tl}\rangle \cdot \mathsf{concatS} \;=\; (\mathsf{id}\times\mathsf{concatS}) \cdot \langle \mathsf{hd}\cdot\pi_1, \mathsf{s}\cdot(\mathsf{tl}\times\mathsf{id})\rangle \\
\equiv \quad & \{\ \times\text{-fusão e absorção}\} \\
& \langle \mathsf{hd}\cdot\mathsf{concatS}, \mathsf{tl}\cdot\mathsf{concatS}\rangle \;=\; \langle \mathsf{id}\cdot(\mathsf{hd}\cdot\pi_1), \mathsf{concatS}\cdot(\mathsf{s}\cdot(\mathsf{tl}\times\mathsf{id}))\rangle \\
\equiv \quad & \{\ \text{definição de identidade}\} \\
& \langle \mathsf{hd}\cdot\mathsf{concatS}, \mathsf{tl}\cdot\mathsf{concatS}\rangle \;=\; \langle \mathsf{hd}\cdot\pi_1, \mathsf{concatS}\cdot(\mathsf{s}\cdot(\mathsf{tl}\times\mathsf{id}))\rangle \\
\equiv \quad & \{\ \text{definição de split}\} \\
& \mathsf{hd}\cdot\mathsf{concatS} \;=\; \mathsf{hd}\cdot\pi_1 \\
& \mathsf{tl}\cdot\mathsf{concatS} \;=\; \mathsf{concatS}\cdot(\mathsf{s}\cdot(\mathsf{tl}\times\mathsf{id}))
\end{aligned}$$

Portanto, $\mathsf{concatS} = [\![\langle \mathsf{hd}\cdot\pi_1, \mathsf{s}\cdot(\mathsf{tl}\times\mathsf{id})\rangle]\!]$.

5. $\boxed{\mathsf{double} : A^\omega \longleftarrow A^\omega}$
A função double duplica cada um dos elementos da stream.

A definição explícita é:

$$\begin{aligned}
\mathsf{double}(\sigma) &= \sigma(0) \\
\mathsf{double}(\sigma)' &= \sigma(0) : \mathsf{double}(\sigma')
\end{aligned}$$

o que se valida provando:

$$\mathsf{double}(\sigma) = \sigma(0) : \sigma(0) : \mathsf{double}(\sigma')$$

Parece-nos logo óbvia esta igualdade pela duplicação de cada elemento. Assim, podemos dizer que:

$$\begin{aligned}
\mathsf{double}(\sigma)(0) &= \sigma(0)\\
\mathsf{double}(\sigma)' &= \sigma(0) : \mathsf{double}(\sigma')\\
\mathsf{double}(\sigma)'' &= \mathsf{double}(\sigma'')\\
\mathsf{double}(\sigma)''' &= \sigma(1) : \mathsf{double}(\sigma'')
\end{aligned}$$

Definição pela Propriedade Universal
Este é um caso curioso: o gene da função necessita de um mecanismo de controlo para assegurar que cada elemento é duplicado apenas uma vez. Esse mecanismo aqui é um valor booleano inicializado a true e que controla a continuação do processo: a cauda da continuação inclui e exclui alternadamente a cabeça corrente.

Comecemos por representar o diagrama. Note-se a forma de especificar a inicialização:

$$\begin{array}{ccc}
\mathsf{A}^\omega & & \\
{\scriptstyle \langle \mathsf{id},\mathsf{true}\rangle}\downarrow & & \\
\mathsf{A}^\omega \times \mathsf{Bool} & \xrightarrow{\varphi} & \mathsf{A} \times (\mathsf{A}^\omega \times \mathsf{Bool})\\
{\scriptstyle \mathsf{dupS}}\downarrow & & \downarrow{\scriptstyle \mathsf{id}\times\mathsf{dupS}}\\
\mathsf{A}^\omega & \xrightarrow{\langle \mathsf{hd},\mathsf{tl}\rangle} & \mathsf{A} \times \mathsf{A}^\omega
\end{array}$$

onde

$$\varphi = \langle \mathsf{hd} \cdot \pi_1, \pi_2 \to \pi_1 \times \neg \bullet \pi_2, \mathsf{tl} \cdot \pi_1 \times \neg \bullet \pi_2 \rangle$$

Assim,

$$\begin{aligned}
& \langle \mathsf{hd}, \mathsf{tl}\rangle \cdot \mathsf{dupS} \;=\; (\mathsf{id} \times \mathsf{dupS}) \cdot \varphi\\
\equiv \quad & \{ \times\text{-fusão e absorção e pela definição de } \varphi \}\\
& \langle \mathsf{hd} \cdot \mathsf{dupS}, \mathsf{tl} \cdot \mathsf{dupS}\rangle \;=\; \langle \mathsf{hd} \cdot \pi_1, \mathsf{dupS} \cdot \pi_2 \to (\pi_1 \times \neg \bullet \pi_2), (\mathsf{tl} \cdot \pi_1 \times \neg \bullet \pi_2)\rangle\\
\equiv \quad & \{ \text{fusão associada ao condicional (2.29)} \}\\
& \langle \mathsf{hd} \cdot \mathsf{dupS}, \mathsf{tl} \cdot \mathsf{dupS}\rangle \;=\; \langle \mathsf{hd} \cdot \pi_1, \pi_2 \to (\mathsf{dupS} \cdot (\pi_1 \times \neg \bullet \pi_2), \mathsf{dupS} \cdot (\mathsf{tl} \cdot \pi_1 \times \neg \bullet \pi_2)\rangle
\end{aligned}$$

Portanto, $\mathsf{dupS} = [\!(\langle \mathsf{hd} \cdot \pi_1, \pi_2 \to (\pi_1 \times \neg \bullet \pi_2), (\mathsf{tl} \cdot \pi_1 \times \neg \bullet \pi_2)\rangle)\!]$.

6. $\boxed{\mathsf{iterateS[f]} : A^\omega \longleftarrow A}$
A função $\mathsf{iterateS}[f]$ aplica a função $f : A \longleftarrow A$ a um determinado elemento, tantas vezes quanto o número da posição em que irá ficar na stream. Ou seja, na cabeça fica o elemento, sem aplicação da função, na posição 1, aplica-se uma vez a função f, na posição 2, aplica-se duas vezes e assim sucessivamente.

Em [Rut05] define-se

$$\begin{aligned}\mathsf{iterateS}[f](0)(a) &= a\\ \mathsf{iterateS}[f](a)' &= \mathsf{iterateS}[f](f(a))\end{aligned}$$

o que se valida via:

$$\mathsf{iterateS}[f](a) = (a, f(a), f(f(a)), \ldots).$$

$$\begin{aligned}\mathsf{iterateS}[f](a) &= \mathsf{iterateS}[f](a)(0) : \mathsf{iterateS}[f](a)'\\ &= a : \mathsf{iterateS}[f](f(a))\\ &= a : f(a) : \mathsf{iterateS}[f](f(f(a)))\end{aligned}$$

Definição pela Propriedade Universal
Comecemos por representar o diagrama:

$$\begin{array}{ccc} \mathsf{A} & \xrightarrow{\mathsf{iterateS}[f]} & \mathsf{A}^\omega \\ {\scriptstyle \langle \mathsf{id}, f\rangle}\downarrow & & \downarrow{\scriptstyle \langle \mathsf{hd},\mathsf{tl}\rangle} \\ \mathsf{A}\times\mathsf{A} & \xrightarrow{\mathsf{id}\times\mathsf{iterateS}[f]} & \mathsf{A}\times\mathsf{A}^\omega \end{array}$$

Como o diagrama comuta, vem que

$$\begin{aligned} & \langle \mathsf{hd}, \mathsf{tl}\rangle \cdot \mathsf{iterateS}[f] \;=\; (\mathsf{id}\times \mathsf{iterateS}[f]) \cdot \langle \mathsf{id}, f\rangle \\ \equiv \quad & \{\times\text{-fusão e absorção}\} \\ & \langle \mathsf{hd}\cdot \mathsf{iterateS}[f], \mathsf{tl}\cdot \mathsf{iterateS}[f]\rangle \;=\; \langle \mathsf{id}, \mathsf{iterateS}[f]\cdot f\rangle \end{aligned}$$

Portanto, $\mathsf{iterateS}[f] = [\![(\langle \mathsf{id}, f\rangle)]\!]$.

7. $\boxed{\mathsf{mapS[f]} : B^\omega \longleftarrow A^\omega}$
A função $\mathsf{map}[f]$ aplica a função $f : B \longleftarrow A$, apenas uma vez, a cada elemento da stream.

Segundo Rutten tem-se que:

$$\begin{aligned} \mathsf{mapS}[f](\sigma)(0) &= f(\sigma(0)) \\ \mathsf{mapS}[f](\sigma)' &= \mathsf{mapS}[f](\sigma') \end{aligned}$$

para cuja validação se prova a seguinte igualdade:

$$\mathsf{mapS}[f](\sigma) = (f(\sigma(0)), f(\sigma(1)), f(\sigma(2)), \ldots).$$

$$\begin{aligned} \mathsf{mapS}[f](\sigma) &= \mathsf{mapS}[f](\sigma)(0) : \mathsf{mapS}[f](\sigma)' \\ &= f(\sigma(0)) : \mathsf{mapS}[f](\sigma') \\ &= f(\sigma(0)) : \mathsf{mapS}[f](\sigma')(0) : \mathsf{mapS}[f](\sigma')' \\ &= f(\sigma(0)) : f(\sigma(1)) : \mathsf{mapS}[f](\sigma'') \end{aligned}$$

Definição pela Propriedade Universal
Comecemos por representar o diagrama:

$$\begin{array}{ccc} \mathsf{A}^\omega & \xrightarrow{\mathsf{mapS}[f]} & \mathsf{B}^\omega \\ {\scriptstyle \langle f\cdot \mathsf{hd}, \mathsf{tl}\rangle}\downarrow & & \downarrow{\scriptstyle \langle \mathsf{hd}, \mathsf{tl}\rangle} \\ \mathsf{B} \times \mathsf{A}^\omega & \xrightarrow{\mathsf{id}\times\mathsf{mapS}[f]} & \mathsf{B} \times \mathsf{B}^\omega \end{array}$$

Como o diagrama comuta, vem que

$$\begin{aligned} & \langle \mathsf{hd}, \mathsf{tl}\rangle \cdot \mathsf{mapS}[f] \;=\; (\mathsf{id} \times \mathsf{mapS}[f]) \cdot \langle f \cdot \mathsf{hd}, \mathsf{tl}\rangle \\ \equiv \quad & \{\text{ $\times$-fusão e absorção}\} \\ & \langle \mathsf{hd} \cdot \mathsf{mapS}[f], \mathsf{tl} \cdot \mathsf{mapS}[f]\rangle \;=\; \langle f \cdot \mathsf{hd}, \mathsf{mapS}[f] \cdot \mathsf{tl}\rangle \end{aligned}$$

Portanto, $\mathsf{mapS}[f] = [\![\langle f \cdot \mathsf{hd}, \mathsf{tl}\rangle]\!]$.

3.2.3 Provas por Coindução

Nesta subsecção vamos explorar a verificação de propriedades pelos dois estilos de raciocínio coindutivo em estudo. Para cada propriedade, apresentamos um *printout* de uma pequena animação em HASKELL.

1. $\boxed{\mathsf{even}(\mathsf{concatS}(\sigma,\tau)) = \sigma}$

Animação

```
Corecursion> toHList(even(concatS(natS,oneS)))
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,
58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90{Interrupted!}

Corecursion> toHList(natS)
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,
58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90{Interrupted!}

Corecursion> toHList(oneS)
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1{Interrupted!}
```

Note–se que a função `toHList` tem por objectivo converter streams em listas do HASKELL para efeitos de visualização. Portanto, por observação destes resultados, somos levemos a conjecturar que

$$\mathsf{even}(\mathsf{concatS}(natS, oneS)) = natS$$

Coindução por Bissimulação

$$\begin{aligned}
& \mathsf{even}(\mathsf{concatS}(\sigma,\tau)) \\
= & \quad \{ \text{pela definição de Stream} \} \\
& \mathsf{even}(\mathsf{concatS}(\sigma,\tau))(0) : \mathsf{even}(\mathsf{concatS}(\sigma,\tau))' \\
= & \quad \{ \text{pela definição de even} \} \\
& \mathsf{concatS}(\sigma,\tau)(0) : \mathsf{even}(\mathsf{concatS}(\sigma,\tau)'') \\
= & \quad \{ \text{pela definição de concatS} \} \\
& \sigma(0) : \mathsf{even}(\mathsf{concatS}(\sigma',\tau'))
\end{aligned}$$

Vamos comparar $\mathsf{even}(\mathsf{concatS}(\sigma,\tau))$ e σ.
Comparando as cabeças, vem que

$$\mathsf{even}(\mathsf{concatS}(\sigma,\tau))(0) = \sigma(0)$$

que sai directamente.
Comparando as caudas, vem que

$$\mathsf{even}(\mathsf{concatS}(\sigma,\tau))' = \mathsf{even}(\mathsf{concatS}(\sigma',\tau')).$$

Então, consideremos a relação $R \subseteq A^\omega \times A^\omega$:

$$R = \{\langle \mathsf{even}(\mathsf{concatS}(\sigma,\tau)), \sigma\rangle | \sigma,\tau \in A^\omega\}$$

que é uma bissimulação.

Coindução por Cálculo

Lema 1 $\mathsf{even} \cdot \mathsf{concatS} = \pi_1$.

Prova. Para realizar a prova é necessário definir π_1 como um anamorfismo para podermos usar a lei de fusão. Consideremos $\pi_1 = [\![\langle \mathsf{hd}\cdot\pi_1, \mathsf{tl}\times\mathsf{tl}\rangle]\!]$. Pela comutatividade do respectivo diagrama tem-se

$$\begin{array}{ccc} \mathsf{A}^\omega \times \mathsf{B}^\omega & \xrightarrow{\pi_1} & \mathsf{A}^\omega \\ {\scriptstyle \langle \mathsf{hd}\cdot\pi_1, \mathsf{tl}\times\mathsf{tl}\rangle}\downarrow & & \downarrow{\scriptstyle \langle \mathsf{hd},\mathsf{tl}\rangle} \\ \mathsf{A}\times(\mathsf{A}^\omega \times \mathsf{B}^\omega) & \xrightarrow{\mathsf{id}\times\pi_1} & \mathsf{A}\times\mathsf{A}^\omega \end{array}$$

$$\begin{array}{ll} & \langle \mathsf{hd},\mathsf{tl}\rangle \cdot \pi_1 \;=\; (\mathsf{id}\times\pi_1)\cdot\langle \mathsf{hd}\cdot\pi_1, \mathsf{tl}\times\mathsf{tl}\rangle \\ \equiv & \quad \{\times\text{-fusão e absorção}\} \\ & \langle \mathsf{hd}\cdot\pi_1, \mathsf{tl}\cdot\pi_1\rangle \;=\; \langle \mathsf{id}\cdot\mathsf{hd}\cdot\pi_1, \pi_1\cdot(\mathsf{tl}\times\mathsf{tl})\rangle \\ \equiv & \quad \{\text{por definição de identidade e pela igualdade de splits}\} \\ & \mathsf{hd}\cdot\pi_1 \;=\; \mathsf{hd}\cdot\pi_1 \\ & \mathsf{tl}\cdot\pi_1 \;=\; \pi_1\cdot(\mathsf{tl}\times\mathsf{tl}) \end{array}$$

Agora que já definimos π_1 como um anamorfismo, podemos passar à demostração, usando a lei de fusão-ana.

$$
\begin{array}{ll}
 & [\![(\langle \mathsf{hd}, \mathsf{tl} \cdot \mathsf{tl}\rangle)]\!] \cdot \mathsf{concatS} \;=\; [\![(\langle \mathsf{hd} \cdot \pi_1, \mathsf{tl} \times \mathsf{tl}\rangle)]\!] \\
\Leftarrow & \quad \{ \text{lei de fusão-ana} \} \\
 & \langle \mathsf{hd}, \mathsf{tl} \cdot \mathsf{tl}\rangle \cdot \mathsf{concatS} \;=\; (\mathsf{id} \times \mathsf{concatS}) \cdot \langle \mathsf{hd} \cdot \pi_1, \mathsf{tl} \times \mathsf{tl}\rangle \\
\equiv & \quad \{ \times\text{-fusão e absorção} \} \\
 & \langle \mathsf{hd} \cdot \mathsf{concatS}, \mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{concatS}\rangle \;=\; \langle \mathsf{id} \cdot \mathsf{hd} \cdot \pi_1, \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl})\rangle \\
\equiv & \quad \{ \text{pela definição de identidade e pela igualdade de splits} \} \\
 & \mathsf{hd} \cdot \mathsf{concatS} \;=\; \mathsf{hd} \cdot \pi_1 \\
 & \mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{concatS} \;=\; \mathsf{concatS} \cdot \pi_1 \cdot (\mathsf{tl} \times \mathsf{tl}) \\
\equiv & \quad \{ \text{pela definição de } \mathsf{concatS} \} \\
 & \mathsf{hd} \cdot \pi_1 \;=\; \mathsf{hd} \cdot \pi_1 \\
 & \mathsf{tl} \cdot \mathsf{concatS} \cdot \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \\
\equiv & \quad \{ \text{pela definição de } \mathsf{concatS} \} \\
 & \mathsf{hd} \cdot \pi_1 \;=\; \mathsf{hd} \cdot \pi_1 \\
 & \mathsf{concatS} \cdot \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \cdot \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \\
\equiv & \quad \{ \text{por } (\mathsf{tl} \times \mathsf{id}) \cdot \mathsf{s} \;=\; \mathsf{s} \cdot (\mathsf{id} \times \mathsf{tl}) \} \\
 & \mathsf{hd} \cdot \pi_1 \;=\; \mathsf{hd} \cdot \pi_1 \\
 & \mathsf{concatS} \cdot \mathsf{s} \cdot \mathsf{s} \cdot (\mathsf{id} \times \mathsf{tl}) \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \\
\equiv & \quad \{ \text{por s ser isomorfismo} \} \\
 & \mathsf{hd} \cdot \pi_1 \;=\; \mathsf{hd} \cdot \pi_1 \\
 & \mathsf{concatS} \cdot (\mathsf{id} \times \mathsf{tl}) \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \\
\equiv & \quad \{ \times\text{-functor} \} \\
 & \mathsf{hd} \cdot \pi_1 \;=\; \mathsf{hd} \cdot \pi_1 \\
 & \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl})
\end{array}
$$

□

2. $\boxed{\mathsf{concatS}(\mathsf{even}(\sigma), \mathsf{odd}(\sigma)) = \sigma}$

Animação

```
Corecursion> toHList(concatS(even(natS),odd(natS)))
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
```

```
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,
58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90{Interrupted!}
```

Portanto, por observação do Haskell, facilmente conjecturamos que

$$\mathsf{concatS}(\mathsf{even}(natS), \mathsf{odd}(natS)) = natS$$

Coindução por Bissimulação

$$\begin{aligned}
& \mathsf{concatS}(\mathsf{even}(\sigma), \mathsf{odd}(\sigma)) \\
= & \quad \{ \text{pela definição de Stream} \} \\
& \mathsf{concatS}(\mathsf{even}(\sigma), \mathsf{odd}(\sigma))(0) : \mathsf{concatS}(\mathsf{even}(\sigma), \mathsf{odd}(\sigma))' \\
= & \quad \{ \text{pela definição de concatS} \} \\
& \mathsf{even}(\sigma)(0) : \mathsf{concatS}(\mathsf{odd}(\sigma), \mathsf{even}(\sigma)') \\
= & \quad \{ \text{pela definição de even} \} \\
& \sigma(0) : \mathsf{concatS}(\mathsf{odd}(\sigma), \mathsf{even}(\sigma''))
\end{aligned}$$

Vamos comparar $\mathsf{concatS}(\mathsf{even}(\sigma), \mathsf{odd}(\sigma))$ e σ.
Comparando as cabeças, vem que

$\mathsf{concatS}(\mathsf{even}(\sigma), \mathsf{odd}(\sigma))(0) = \sigma(0)$, que sai directamente.

Comparando as caudas, vem que

$$\mathsf{concatS}(\mathsf{even}(\sigma), \mathsf{odd}(\sigma))' = \mathsf{concatS}(\mathsf{odd}(\sigma), \mathsf{even}(\sigma'')).$$

Então, não é suficiente considerarmos a relação $R \subseteq A^\omega \times A^\omega$:

$$R = \{\langle \mathsf{concatS}(\mathsf{even}(\sigma), \mathsf{odd}(\sigma)), \sigma \rangle | \sigma \in A^\omega\}.$$

Consideremos, assim, a relação:

$$\begin{aligned}
R \ = & \{\langle \mathsf{concatS}(\mathsf{even}(\sigma), \mathsf{odd}(\sigma)), \sigma \rangle | \ \sigma \in A^\omega\} \\
& \cup \{\langle \mathsf{concatS}(\mathsf{odd}(\sigma), \mathsf{even}(\sigma'')), \sigma' \rangle | \ \sigma \in A^\omega\}
\end{aligned}$$

Vamos provar que a segunda também é bissimulação e assim, prova-se que R é ainda bissimulação, uma vez que R é a união de duas bissimulações.

$$
\begin{array}{ll}
 & \mathsf{concatS}(\mathsf{odd}(\sigma), \mathsf{even}(\sigma'')) \\
= & \quad \{ \text{pela definição de Stream} \} \\
 & \mathsf{concatS}(\mathsf{odd}(\sigma), \mathsf{even}(\sigma''))(0) : \mathsf{concatS}(\mathsf{odd}(\sigma), \mathsf{even}(\sigma''))' \\
= & \quad \{ \text{pela definição de concatS} \} \\
 & \mathsf{odd}(\sigma)(0) : \mathsf{concatS}(\mathsf{even}(\sigma''), \mathsf{odd}(\sigma)') \\
= & \quad \{ \text{pela definição de odd} \} \\
 & \sigma(1) : \mathsf{concatS}(\mathsf{even}(\sigma''), \mathsf{odd}(\sigma''))
\end{array}
$$

(i) $\mathsf{concatS}(\mathsf{odd}(\sigma), \mathsf{even}(\sigma''))(0) = \sigma(1) = \sigma'(0)$

(ii) $\mathsf{concatS}(\mathsf{odd}(\sigma), \mathsf{even}(\sigma''))' = \mathsf{concatS}(\mathsf{even}(\sigma''), \mathsf{odd}(\sigma''))$
$\in \{\langle \mathsf{concatS}(\mathsf{even}(\sigma), \mathsf{odd}(\sigma)), \sigma \rangle | \sigma \in A^{\omega}\} \subseteq R$

Logo, fica provado que R é uma bisssimulação.

Coindução por Cálculo

O que se pretende mostrar é que

Lema 2 $\mathsf{concatS} \cdot \langle \mathsf{even}, \mathsf{odd} \rangle = \mathsf{id}$.

Prova. Para isso, vamos ter que definir a identidade como um anamorfismo para podermos usar a lei de fusão.
Consideremos $\mathsf{id} = [\![(\langle \mathsf{hd}, \mathsf{tl} \rangle)]\!]$.

$$
\begin{array}{ll}
 & [\![(\langle \mathsf{hd} \cdot \pi_1, \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \rangle)]\!] \cdot \langle \mathsf{even}, \mathsf{odd} \rangle \;=\; [\![(\langle \mathsf{hd}, \mathsf{tl} \rangle)]\!] \\
\Leftarrow & \quad \{ \text{lei de fusão-ana} \} \\
 & \langle \mathsf{hd} \cdot \pi_1, \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \rangle \cdot \langle \mathsf{even}, \mathsf{odd} \rangle \;=\; (\mathsf{id} \times \langle \mathsf{even}, \mathsf{odd} \rangle) \cdot \langle \mathsf{hd}, \mathsf{tl} \rangle \\
\equiv & \quad \{ \times\text{-fusão e absorção} \} \\
 & \langle \mathsf{hd} \cdot \pi_1 \cdot \langle \mathsf{even}, \mathsf{odd} \rangle, \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \cdot \langle \mathsf{even}, \mathsf{odd} \rangle \rangle \;=\; \langle \mathsf{id} \cdot \mathsf{hd}, \langle \mathsf{even}, \mathsf{odd} \rangle \cdot \mathsf{tl} \rangle \\
\equiv & \quad \{ \text{lei de cancelamento-}\times\text{, por absorção-}\times \text{ e pela definição de identidade} \} \\
 & \langle \mathsf{hd} \cdot \mathsf{even}, \mathsf{s} \langle \mathsf{tl} \cdot \mathsf{even}, \mathsf{id} \cdot \mathsf{odd} \rangle \rangle \;=\; \langle \mathsf{hd}, \langle \mathsf{even} \cdot \mathsf{tl}, \mathsf{odd} \cdot \mathsf{tl} \rangle \rangle \\
\equiv & \quad \{ \text{por } \mathsf{s} = \langle \pi_2, \pi_1 \rangle \} \\
 & \langle \mathsf{hd} \cdot \mathsf{even}, \langle \pi_2, \pi_1 \rangle \cdot \langle \mathsf{tl} \cdot \mathsf{even}, \mathsf{odd} \rangle \rangle \;=\; \langle \mathsf{hd}, \langle \mathsf{even} \cdot \mathsf{tl}, \mathsf{odd} \cdot \mathsf{tl} \rangle \rangle \\
\equiv & \quad \{ \text{lei de cancelamento-}\times \} \\
 & \langle \mathsf{hd} \cdot \mathsf{even}, \langle \mathsf{odd}, \mathsf{tl} \cdot \mathsf{even} \rangle \rangle \;=\; \langle \mathsf{hd}, \langle \mathsf{even} \cdot \mathsf{tl}, \mathsf{odd} \cdot \mathsf{tl} \rangle \rangle \\
\equiv & \quad \{ \text{pela igualdade de splits} \} \\
 & \mathsf{hd} \cdot \mathsf{even} \;=\; \mathsf{hd}
\end{array}
$$

$$
\begin{array}{ll}
& \langle \mathsf{odd}, \mathsf{tl} \cdot \mathsf{even} \rangle \;=\; \langle \mathsf{even} \cdot \mathsf{tl}, \mathsf{odd} \cdot \mathsf{tl} \rangle \\
\equiv & \quad \{ \text{pela igualdade de splits} \} \\
& \mathsf{hd} \cdot \mathsf{even} \;=\; \mathsf{hd} \\
& \mathsf{odd} \;=\; \mathsf{even} \cdot \mathsf{tl} \\
& \mathsf{tl} \cdot \mathsf{even} \;=\; \mathsf{odd} \cdot \mathsf{tl}
\end{array}
$$

Agora temos que provar que as igualdades anteriores são verdadeiras. A primeira igualdade vem directamente da definição de even.
As restantes igualdades necessitam de ser provadas.

Provemos,

$$\mathsf{odd} \;=\; \mathsf{even} \cdot \mathsf{tl} \tag{3.7}$$

Para isso, consideremos a definição de odd e even como anamorfismo e usemos a lei de fusão-ana. Vamos reescrevê-la:

$$
\begin{array}{ll}
& [\![(\langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \cdot \mathsf{tl} \rangle)]\!] \;=\; [\![(\langle \mathsf{hd}, \mathsf{tl} \cdot \mathsf{tl} \rangle)]\!] \cdot \mathsf{tl} \\
\Leftarrow & \quad \{ \text{lei de fusão-ana} \} \\
& \langle \mathsf{hd}, \mathsf{tl} \cdot \mathsf{tl} \rangle \cdot \mathsf{tl} \;=\; (\mathsf{id} \times \mathsf{tl}) \cdot \langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \cdot \mathsf{tl} \rangle \\
\equiv & \quad \{ \times\text{-fusão e absorção} \} \\
& \langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{tl} \rangle \;=\; \langle \mathsf{id} \cdot \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{tl} \rangle \\
\equiv & \quad \{ \text{pela igualdade de splits e pela definição de identidade} \} \\
& \mathsf{hd} \cdot \mathsf{tl} \;=\; \mathsf{hd} \cdot \mathsf{tl} \\
& \mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{tl} \;=\; \mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{tl}
\end{array}
$$

Quanto à igualdade $\mathsf{tl} \cdot \mathsf{even} \;=\; \mathsf{odd} \cdot \mathsf{tl}$, esta vem da definição de even, substituindo $\mathsf{tl} \cdot \mathsf{even}$ por $\mathsf{even} \cdot \mathsf{tl} \cdot \mathsf{tl}$. Mas, como pela igualdade (3.7) $\mathsf{odd} \;=\; \mathsf{even} \cdot \mathsf{tl}$, basta substituir e encontramos $\mathsf{odd} \cdot \mathsf{tl} \;=\; \mathsf{odd} \cdot \mathsf{tl}$.
Provadas que estão as três igualdades, concluímos pela lei fusão-ana que:

$$[\![(\langle \mathsf{hd} \cdot \pi_1, \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \rangle)]\!] \cdot \langle \mathsf{even}, \mathsf{odd} \rangle \;=\; [\![(\langle \mathsf{hd}, \mathsf{tl} \rangle)]\!]$$

□

3. $\boxed{\mathsf{odd}(\mathsf{concatS}(\sigma, \tau)) = \tau}$

Animação

```
Corecursion> toHList(odd(concatS(natS,oneS)))
[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1{Interrupted!}
```

o que sustenta a suspeita de que

$$\mathsf{odd}(\mathsf{concatS}(\mathsf{natS}, \mathsf{oneS}) \;=\; \mathsf{one}S\,.$$

Coindução por Bissimulação

$$\begin{array}{ll} & \mathsf{odd}(\mathsf{concatS}(\sigma,\tau)) \\ = & \quad \{\text{pela definição de Stream}\} \\ & \mathsf{odd}(\mathsf{concatS}(\sigma,\tau))(0) : \mathsf{odd}(\mathsf{concatS}(\sigma,\tau))' \\ = & \quad \{\text{pela definição de odd}\} \\ & \mathsf{concatS}(\sigma,\tau)(1) : \mathsf{odd}(\mathsf{concatS}(\sigma,\tau)'') \\ = & \quad \{\text{pela definição de concatS}\} \\ & \tau(0) : \mathsf{odd}(\mathsf{concatS}(\sigma',\tau')) \end{array}$$

Vamos comparar $\mathsf{odd}(\mathsf{concatS}(\sigma,\tau))$ e τ.
Comparando as cabeças, vem que

$$\mathsf{odd}(\mathsf{concatS}(\sigma,\tau))(0) = \tau(0), \text{ que sai directamente.}$$

Comparando as caudas, vem que

$$\mathsf{odd}(\mathsf{concatS}(\sigma,\tau))' = \mathsf{odd}(\mathsf{concatS}(\sigma',\tau')).$$

Então, é suficiente considerarmos a relação $R \subseteq A^\omega \times A^\omega$:

$$R = \{\langle \mathsf{odd}(\mathsf{concatS}(\sigma,\tau)), \tau\rangle | \sigma \in A^\omega\}$$

que é uma bissimulação.

Coindução por Cálculo

A prova faz-se de forma similar à demonstração que $\mathsf{even} \cdot \mathsf{concatS} = \pi_1$,

Lema 3 $\mathsf{odd} \cdot \mathsf{concatS} = \pi_2$

Prova. Vamos começar por definir π_2 como um anamorfismo. Consideremos $\pi_2 = [\![(\langle \mathsf{hd} \cdot \pi_2, \mathsf{tl} \times \mathsf{tl} \rangle)]\!]$. Observe-se o respectivo diagrama:

$$\begin{array}{ccc} \mathsf{A}^\omega \times \mathsf{B}^\omega & \xrightarrow{\pi_2} & \mathsf{B}^\omega \\ {\scriptstyle \langle \mathsf{hd}\cdot\pi_2, \mathsf{tl}\times\mathsf{tl}\rangle}\downarrow & & \downarrow{\scriptstyle \langle \mathsf{hd},\mathsf{tl}\rangle} \\ \mathsf{B} \times (\mathsf{A}^\omega \times \mathsf{B}^\omega) & \xrightarrow{\mathsf{id}\times\pi_2} & \mathsf{B} \times \mathsf{B}^\omega \end{array}$$

Como este diagrama comuta, tem-se que:

$$\begin{array}{ll} & \langle \mathsf{hd}, \mathsf{tl} \rangle \cdot \pi_2 \;=\; (\mathsf{id} \times \pi_2) \cdot \langle \mathsf{hd} \cdot \pi_2, \mathsf{tl} \times \mathsf{tl} \rangle \\ \equiv & \quad \{ \times\text{-fusão e absorção} \} \\ & \langle \mathsf{hd} \cdot \pi_2, \mathsf{tl} \cdot \pi_2 \rangle \;=\; \langle \mathsf{id} \cdot \mathsf{hd} \cdot \pi_2, \pi_2 \cdot (\mathsf{tl} \times \mathsf{tl}) \rangle \\ \equiv & \quad \{ \text{por definição de identidade e pela igualdade de splits} \} \\ & \mathsf{hd} \cdot \pi_2 \;=\; \mathsf{hd} \cdot \pi_2 \\ & \mathsf{tl} \cdot \pi_2 \;=\; \pi_2 \cdot (\mathsf{tl} \times \mathsf{tl}) \end{array}$$

Agora que já definimos π_2 como um anamorfismo, podemos passar à demostração, usando a lei de fusão-ana.

$$\begin{array}{ll} & [\![(\langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \cdot \mathsf{tl} \rangle)]\!] \cdot \mathsf{concatS} \;=\; [\![(\langle \mathsf{hd} \cdot \pi_2, \mathsf{tl} \times \mathsf{tl} \rangle)]\!] \\ \Leftarrow & \quad \{ \text{lei de fusão-ana} \} \\ & \langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \cdot \mathsf{tl} \rangle \cdot \mathsf{concatS} \;=\; (\mathsf{id} \times \mathsf{concatS}) \cdot \langle \mathsf{hd} \cdot \pi_2, \mathsf{tl} \times \mathsf{tl} \rangle \\ \equiv & \quad \{ \times\text{-fusão e absorção} \} \\ & \langle \mathsf{hd} \cdot \mathsf{tl} \cdot \mathsf{concatS}, \mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{concatS} \rangle \;=\; \langle \mathsf{id} \cdot \mathsf{hd} \cdot \pi_2, \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \rangle \\ \equiv & \quad \{ \text{pela definição de identidade e pela igualdade de splits} \} \\ & \mathsf{hd} \cdot \mathsf{tl} \cdot \mathsf{concatS} \;=\; \mathsf{hd} \cdot \pi_2 \\ & \mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{concatS} \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \\ \equiv & \quad \{ \text{pela definição de concatS} \} \\ & \mathsf{hd} \cdot \mathsf{concatS} \cdot \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{hd} \cdot \pi_2 \\ & \mathsf{tl} \cdot \mathsf{concatS} \cdot \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \\ \equiv & \quad \{ \text{pela definição de concatS} \} \\ & \mathsf{hd} \cdot \pi_1 \cdot \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{hd} \cdot \pi_2 \\ & \mathsf{concatS} \cdot \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \cdot \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \\ \equiv & \quad \{ \text{por } (\mathsf{tl} \times \mathsf{id}) \cdot \mathsf{s} \;=\; \mathsf{s} \cdot (\mathsf{id} \times \mathsf{tl}) \text{ e pela definição de s} \} \end{array}$$

$$
\begin{array}{ll}
 & \mathsf{hd} \cdot \pi_1 \cdot \langle \pi_2, \pi_1 \rangle \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{hd} \cdot \pi_2 \\
 & \mathsf{concatS} \cdot \mathsf{s} \cdot \mathsf{s} \cdot (\mathsf{id} \times \mathsf{tl}) \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \\
\equiv & \quad \{ \text{por s ser isomorfismo e por } \pi_1 \cdot \langle \pi_2, \pi_1 \rangle = \pi_2 \} \\
 & \mathsf{hd} \cdot \pi_2 \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{hd} \cdot \pi_2 \\
 & \mathsf{concatS} \cdot (\mathsf{id} \times \mathsf{tl}) \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \\
\equiv & \quad \{ \times\text{-fuctor} \} \\
 & \mathsf{hd} \cdot \pi_2 \cdot (\mathsf{tl} \times \mathsf{id}) \;=\; \mathsf{hd} \cdot \pi_2 \\
 & \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \\
\equiv & \quad \{ \text{por } \mathsf{tl} \times \mathsf{tl} = \langle \mathsf{tl} \cdot \pi_1, \mathsf{id} \cdot \pi_2 \rangle \} \\
 & \mathsf{hd} \cdot \pi_2 \cdot \langle \mathsf{tl} \cdot \pi_1, \mathsf{id} \cdot \pi_2 \rangle \;=\; \mathsf{hd} \cdot \pi_2 \\
 & \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \\
\equiv & \quad \{ \text{por definição de } \pi_2 \text{ e de } \mathsf{id} \} \\
 & \mathsf{hd} \cdot \pi_2 \;=\; \mathsf{hd} \cdot \pi_2 \\
 & \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl}) \;=\; \mathsf{concatS} \cdot (\mathsf{tl} \times \mathsf{tl})
\end{array}
$$

□

4. $\boxed{\mathsf{mapS}[g] \cdot \mathsf{mapS}[f] = \mathsf{mapS}[g \cdot f]}$

Animação

```
Corecursion> toHList(mapS(2*)oneS)
[2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
,2,2,2,2,2{Interrupted!}

Corecursion> toHList(mapS(3*)oneS)
[3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3
,3,3,3,3,3{Interrupted!}

Corecursion> toHList((mapS(2*).mapS(3*))oneS)
[6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
```

```
,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
,6,6,6,6,6{Interrupted!}

Corecursion> toHList(mapS((2*).(3*))oneS)
[6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
,6,6,6,6,6{Interrupted!}
```

Coindução por Bissimulação

$$
\begin{aligned}
& (\mathsf{mapS[g]} \cdot \mathsf{mapS[f]})\sigma \\
= & \quad \{ \text{pela definição de composição} \} \\
& \mathsf{mapS[g]}(\mathsf{mapS[f]}(\sigma)) \\
= & \quad \{ \text{pela definição de mapS[f]} \} \\
& \mathsf{mapS[g]}(\mathsf{f}(\sigma(0)) : \mathsf{f}(\sigma(1)) : \mathsf{mapS[f]}(\sigma'')) \\
= & \quad \{ \text{pela definição de mapS[g]} \} \\
& \mathsf{g}(\mathsf{f}(\sigma(0))) : \mathsf{g}(\mathsf{f}(\sigma(1))) : \mathsf{g}(\mathsf{mapS[f]}(\sigma'')) \\
= & \quad \{ \text{pela definição de composição de funções} \} \\
& \mathsf{g} \cdot \mathsf{f}(\sigma(0)) : \mathsf{g} \cdot \mathsf{f}(\sigma(1)) : \mathsf{g} \cdot \mathsf{mapS[f]}(\sigma'') \\
= & \quad \{ \text{pela definição de mapS[g · f]} \} \\
& \mathsf{mapS[g \cdot f]}\sigma
\end{aligned}
$$

Então, basta considerarmos a relação $R \subseteq A^\omega \times A^\omega$:

$$R = \{\langle (\mathsf{mapS[g]} \cdot \mathsf{mapS[f]})\sigma, \mathsf{mapS[g \cdot f]}\sigma \rangle | \sigma \in A^\omega\}.$$

Agora estamos em condições de concluir que, quer a cabeça, quer a cauda são iguais e, portanto, esta relação é uma bissimulação.

Coindução por Cálculo

Lema 4 $\mathsf{mapS[g]} \cdot \mathsf{mapS[f]} = \mathsf{mapS[g \cdot f]}$

Prova. Pela definição de mapS, podemos reescrever $\mathsf{mapS[g \cdot f]}$ como $[\![\langle g \cdot f \cdot \mathsf{hd}, \mathsf{tl} \rangle]\!]$. Estamos agora em condições de usar a lei de fusão-ana.

Vejamos:

$$
\begin{array}{ll}
 & [\!(\langle g \cdot \mathsf{hd}, \mathsf{tl}\rangle)\!] \cdot \mathsf{mapS[f]} \;=\; [\!(\langle g \cdot f \cdot \mathsf{hd}, \mathsf{tl}\rangle)\!] \\
\Leftarrow & \quad \{\text{lei de fusão-ana}\} \\
 & \langle g \cdot \mathsf{hd}, \mathsf{tl}\rangle \cdot \mathsf{mapS[f]} \;=\; (\mathsf{id} \times \mathsf{mapS[f]}) \cdot \langle g \cdot f \cdot \mathsf{hd}, \mathsf{tl}\rangle \\
\equiv & \quad \{\times\text{-fusão e absorção}\} \\
 & \langle g \cdot \mathsf{hd} \cdot \mathsf{mapS[f]}, \mathsf{tl} \cdot \mathsf{mapS[f]}\rangle \;=\; \langle \mathsf{id} \cdot g \cdot f \cdot \mathsf{hd}, \mathsf{mapS[f]} \cdot \mathsf{tl}\rangle \\
\equiv & \quad \{\text{pela igualdade de splits e pela definição de identidade}\} \\
 & g \cdot \mathsf{hd} \cdot \mathsf{mapS[f]} = g \cdot f \cdot \mathsf{hd} \\
 & \mathsf{tl} \cdot \mathsf{mapS[f]} = \mathsf{mapS[f]} \cdot \mathsf{tl} \\
\equiv & \quad \{\text{pela definição de mapS}\} \\
 & g \cdot f \cdot \mathsf{hd} = g \cdot f \cdot \mathsf{hd} \\
 & \mathsf{mapS[f]} \cdot \mathsf{tl} = \mathsf{mapS[f]} \cdot \mathsf{tl}
\end{array}
$$

□

5. $\boxed{\mathsf{mapS[f]} \cdot \mathsf{iterateS[f]} = \mathsf{iterateS[f]} \cdot \mathsf{f}}$

Animação

```
Corecursion> toHList(mapS(1+)((iterateS(1+)(0))))
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,
58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90{Interrupted!}

Corecursion> toHList(iterateS(1+)((1+)(0)))
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,
58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90{Interrupted!}
```

Coindução por Bissimulação

$$
\begin{array}{ll}
 & \mathsf{mapS[f]} \cdot \mathsf{iterateS[f]}a \\
= & \quad \{ \text{pela definição de composição} \} \\
 & \mathsf{mapS[f]}(\mathsf{iterateS[f]}(a)) \\
= & \quad \{ \text{pela definição de iterateS[f]} \} \\
 & \mathsf{mapS[f]}(a : \mathsf{f}(a) : \mathsf{iterateS[f]}(\mathsf{f}(\mathsf{f}(a)))) \\
= & \quad \{ \text{pela definição de mapS[f]} \} \\
 & \mathsf{f}(a) : \mathsf{f}(\mathsf{f}(a)) : \mathsf{mapS[f]}(\mathsf{iterateS[f]}(\mathsf{f}(\mathsf{f}(a))))
\end{array}
$$

Agora vamos comparar com o segundo membro.

$$
\begin{array}{ll}
 & \mathsf{iterateS[f]} \cdot \mathsf{f}a \\
= & \quad \{ \text{pela aplicação da função f} \} \\
 & \mathsf{iterateS[f]}(\mathsf{f}(a)) \\
= & \quad \{ \text{pela definição de iterateS} \} \\
 & \mathsf{f}(a) : \mathsf{f}(\mathsf{f}(a)) : \mathsf{iterateS[f]}(\mathsf{f}(\mathsf{f}(\mathsf{f}(a))))
\end{array}
$$

Se observarmos o desenvolvimento do primeiro e do segundo membros, concluímos que é necessário recorrer à indução para demonstrar o resultado. Assim sendo, basta considerarmos a relação $R \subseteq A^\omega \times A^\omega$:

$$R = \{\langle \mathsf{mapS[f]} \cdot \mathsf{iterateS[f]}a, \mathsf{iterateS[f]} \cdot \mathsf{f}a\rangle | a \in A\}.$$

Por indução conclui-se que, quer a cabeça, quer a cauda são iguais e, portanto, esta relação é uma bissimulação.

Coindução por Cálculo

Lema 5 $\mathsf{mapS[f]} \cdot \mathsf{iterateS[f]} = \mathsf{iterateS[f]} \cdot \mathsf{f}$

Prova. Para podermos aplicar as leis associadas ao anamorfismo é necessário exprimir $\mathsf{iterateS[f]} \cdot \mathsf{f}$ como um anamorfismo para A^ω:

$$
\begin{array}{ccc}
A^\omega & \xrightarrow{\langle \mathsf{hd},\mathsf{tl}\rangle} & A \times A^\omega \\
{\scriptstyle \mathsf{iterateS[f]}}\uparrow & & \uparrow{\scriptstyle \mathsf{id}\times\mathsf{iterateS[f]}} \\
A & \xrightarrow{\langle \mathsf{id}, f\rangle} & A \times A \\
{\scriptstyle f}\uparrow & & \uparrow{\scriptstyle \mathsf{id}\times f} \\
A & \xrightarrow{\langle f, f\rangle} & A \times A
\end{array}
$$

Como o diagrama inferior comuta, pois

$$\begin{array}{ll} & \mathsf{id} \times f \cdot \langle f, f \rangle \;=\; \langle \mathsf{id}, f \rangle \cdot f \\ \equiv & \quad \{ \times\text{-absorção e fusão} \} \\ & \langle f, f \cdot f \rangle \;=\; \langle f, f \cdot f \rangle \end{array}$$

vem que,

$$\mathsf{iterate[f]} \cdot f = [\![\langle f, f \rangle]\!]$$

Estamos agora em condições de usar a lei de fusão-ana. Vejamos:

$$\begin{array}{ll} & \mathsf{mapS[f]} \cdot \mathsf{iterate[f]} \;=\; [\![\langle f, f \rangle]\!] \\ \equiv & \quad \{ \text{pela definição de map[f]} \} \\ & [\![\langle f \cdot \mathsf{hd}, \mathsf{tl} \rangle]\!] \cdot \mathsf{iterate[f]} \;=\; [\![\langle f, f \rangle]\!] \\ \Leftarrow & \quad \{ \text{lei de fusão-ana} \} \\ & \langle f \cdot \mathsf{hd}, \mathsf{tl} \rangle \cdot \mathsf{iterateS[f]} \;=\; (\mathsf{id} \times \mathsf{iterateS[f]}) \cdot \langle f, f \rangle \\ \equiv & \quad \{ \times\text{-fusão e absorção} \} \\ & \langle f \cdot \mathsf{hd} \cdot \mathsf{iterateS[f]}, \mathsf{tl} \cdot \mathsf{iterateS[f]} \rangle \;=\; \langle \mathsf{id} \cdot f, \mathsf{iterateS[f]} \cdot \mathsf{f} \rangle \\ \equiv & \quad \{ \text{pela igualdade de splits e pela definição de identidade} \} \\ & f \cdot \mathsf{hd} \cdot \mathsf{iterateS[f]} = f \\ & \mathsf{tl} \cdot \mathsf{iterateS[f]} = \mathsf{iterateS[f]} \cdot f \\ \equiv & \quad \{ \text{pela definição de iterateS[f]} \} \\ & \mathsf{true} \end{array}$$

□

3.2.4 Um Exercício sobre $\mathbb{R}^\omega$

Nesta subsecção vamos prosseguir o estudo da prova por coindução por cálculo. Para isso usaremos alguns exemplos propostos em [Rut05]. Usando a notação da referência citada, vamos considerar duas streams de números reais, σ e τ, denotadas como:

$$\sigma = (\sigma_0, \sigma_1, \sigma_2, \ldots), \tau = (\tau_0, \tau_1, \tau_2, \ldots).$$

Em [Rut05] define-se a soma entre duas streams, somando ordenadamente elemento a elemento, ou seja,

$$\sigma + \tau = (\sigma_0 + \tau_0, \sigma_1 + \tau_1, \sigma_2 + \tau_2, \ldots)$$

Note-se que o mesmo símbolo (+) é usado para a soma de duas streams, bem como, para a soma de dois números reais. Vamos agora definir a soma de streams como um anamorfismo. Assim,

$$\begin{array}{ccc} \mathbb{R}^\omega & \xrightarrow{\langle \mathsf{hd},\mathsf{tl}\rangle} & \mathbb{R} \times \mathbb{R}^\omega \\ {\scriptstyle +}\uparrow & & \uparrow{\scriptstyle \mathsf{id}\times +} \\ \mathbb{R}^\omega \times \mathbb{R}^\omega & \xrightarrow{\langle +\cdot(\mathsf{hd}\times\mathsf{hd}),\mathsf{tl}\times\mathsf{tl}\rangle} & \mathbb{R} \times (\mathbb{R}^\omega \times \mathbb{R}^\omega) \end{array}$$

i.e.,

$$+ \;=\; [\!(\langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl}\rangle)\!] \tag{3.8}$$

Consideremos a seguinte operação sobre *streams*:

$$\begin{array}{ccc} \mathbb{R}^\omega & \xrightarrow{\langle \mathsf{hd},\mathsf{tl}\rangle} & \mathbb{R} \times \mathbb{R}^\omega \\ {\scriptstyle [\,]}\uparrow & & \uparrow{\scriptstyle \mathsf{id}\times[\,]} \\ \mathbb{R} & \xrightarrow{\langle \mathsf{id},\underline{0}\cdot !\rangle} & \mathbb{R} \times \mathbb{R} \end{array}$$

i.e.,

$$[\,] \;=\; [\!(\langle \mathsf{id}, \underline{0}\cdot !\rangle)\!] \tag{3.9}$$

donde

$$[r] = [\,]\, r \quad \text{ou, equivalentemente} \quad \underline{[r]} = [\,] \cdot \underline{r} \tag{3.10}$$

A comutatividade do diagrama acima corresponde à igualdade

$$\langle \mathsf{hd}, \mathsf{tl}\rangle \cdot [\,] \;=\; (\mathsf{id} \times [\,]) \cdot \langle \mathsf{id}, \underline{0}\cdot !\rangle \tag{3.11}$$

donde se deduz

$$\mathsf{hd}\,[r] \;=\; r \quad \wedge \quad \mathsf{tl}\,[r] \;=\; [0] \tag{3.12}$$

porque

$$
\begin{array}{ll}
 & \langle \mathsf{hd}, \mathsf{tl}\rangle \cdot [\,] \;=\; (\mathsf{id} \times [\,]) \cdot \langle \mathsf{id}, \underline{0}\cdot !\rangle \\
\equiv & \quad \{ \text{ leis fusão-produto e absorção-produto } \} \\
 & \langle \mathsf{hd} \cdot [\,], \mathsf{tl} \cdot [\,]\rangle \;=\; \langle \mathsf{id}, [\,] \cdot \underline{0}\cdot !\rangle \\
\equiv & \quad \{ \text{ igualdade de } \textit{splits} \} \\
 & \mathsf{hd} \cdot [\,] \;=\; \mathsf{id} \quad \wedge \quad \mathsf{tl} \cdot [\,] \;=\; [\,] \cdot \underline{0}\cdot ! \\
\Rightarrow & \quad \{ \text{ aplicando a um valor } r \text{ arbitrário} \} \\
 & \mathsf{hd}\,[r] \;=\; r \quad \wedge \quad \mathsf{tl}\,[r] \;=\; ([\,] \cdot \underline{0}\cdot !)\, r \;=\; [\,]\,0 \;=\; [0]
\end{array}
$$

Esta operação vai-nos permitir somar ou multiplicar um número real por uma stream, transformando um número real numa stream que tem à cabeça esse número real e toda a cauda constituída por zeros.

Lema 6

$$\sigma + [0] \;=\; \sigma \tag{3.13}$$

Prova. Em notação *pointfree* a igualdade (3.13) é re-escrita como

$$+ \cdot \langle \mathsf{id}, \underline{[0]}\rangle \;=\; \mathsf{id} \tag{3.14}$$

onde, como habitualmente, $\underline{[0]} = [0]\cdot !$. Donde

$$
\begin{array}{ll}
 & + \cdot \langle \mathsf{id}, \underline{[0]}\rangle \;=\; \mathsf{id} \\
\equiv & \quad \{ \text{ definição (3.8) e lei reflexão-ana} \} \\
 & [\!(\langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl}\rangle)\!] \cdot \langle \mathsf{id}, \underline{[0]}\rangle \;=\; [\!(\langle \mathsf{hd}, \mathsf{tl}\rangle)\!] \\
\Leftarrow & \quad \{ \text{ lei fusão-ana} \} \\
 & \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl}\rangle \cdot \langle \mathsf{id}, \underline{[0]}\rangle \;=\; (\mathsf{id} \times \langle \mathsf{id}, \underline{[0]}\rangle) \cdot \langle \mathsf{hd}, \mathsf{tl}\rangle \\
\equiv & \quad \{ \text{ leis fusão-produto e absorção-produto} \} \\
 & \langle + \cdot (\mathsf{hd} \times \mathsf{hd}) \cdot \langle \mathsf{id}, \underline{[0]}\rangle, (\mathsf{tl} \times \mathsf{tl}) \cdot \langle \mathsf{id}, \underline{[0]}\rangle\rangle \;=\; \langle \mathsf{hd}, \langle \mathsf{id}, \underline{[0]}\rangle \cdot \mathsf{tl}\rangle \\
\equiv & \quad \{ \text{ leis absorção-produto e fusão-produto} \} \\
 & \langle + \cdot \langle \mathsf{hd}, \mathsf{hd} \cdot \underline{[0]}\rangle, \langle \mathsf{tl}, \mathsf{tl} \cdot \underline{[0]}\rangle\rangle \;=\; \langle \mathsf{hd}, \langle \mathsf{tl}, \underline{[0]} \cdot \mathsf{tl}\rangle\rangle \\
\equiv & \quad \{ \text{ definição de ponto } (f \cdot \underline{p} = \underline{f\, p}) \text{ e definição de constante} \} \\
 & \langle + \cdot \langle \mathsf{hd}, \underline{\mathsf{hd}\ [0]}\rangle, \langle \mathsf{tl}, \underline{\mathsf{tl}\ [0]}\rangle\rangle \;=\; \langle \mathsf{hd}, \langle \mathsf{tl}, [0]\cdot ! \cdot \mathsf{tl}\rangle\rangle \\
\equiv & \quad \{ \text{ leis (3.12) e !-universal } (! = ! \cdot f) \} \\
 & \langle + \cdot \langle \mathsf{hd}, \underline{0}\rangle, \langle \mathsf{tl}, \underline{[0]}\rangle\rangle \;=\; \langle \mathsf{hd}, \langle \mathsf{tl}, \underline{[0]}\rangle\rangle \\
\equiv & \quad \{ \text{ aritmética} \}
\end{array}
$$

$$\begin{array}{ll} & \langle \mathsf{hd}, \langle \mathsf{tl}, \underline{[0]} \rangle \rangle \;=\; \langle \mathsf{hd}, \langle \mathsf{tl}, \underline{[0]} \rangle \rangle \\ \equiv & \quad \{ \text{ igualdade de } \textit{splits} \} \\ & \mathsf{true} \end{array}$$

□

Lema 7

$$\sigma + \tau \;=\; \tau + \sigma \tag{3.15}$$

Prova. Em notação *pointfree* a igualdade (3.15) é re-escrita como

$$+ \cdot \mathsf{s} \;=\; + \tag{3.16}$$

onde $\mathsf{s} = \langle \pi_2, \pi_1 \rangle$ é um isomorfismo natural. Donde

$$\begin{array}{ll} & + \cdot \mathsf{s} \;=\; + \\ \equiv & \quad \{ \text{ definição (3.8)} \} \\ & [\![\langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl} \rangle]\!] \cdot \mathsf{s} \;=\; [\![\langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl} \rangle]\!] \\ \Leftarrow & \quad \{ \text{ lei fusão-ana} \} \\ & \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl} \rangle \cdot \mathsf{s} \;=\; (\mathsf{id} \times \mathsf{s}) \cdot \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl} \rangle \\ \equiv & \quad \{ \text{ leis fusão-produto e absorção-produto} \} \\ & \langle + \cdot (\mathsf{hd} \times \mathsf{hd}) \cdot \mathsf{s}, (\mathsf{tl} \times \mathsf{tl}) \cdot \mathsf{s} \rangle \;=\; \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{tl}) \rangle \\ \equiv & \quad \{ \text{ naturalidade de s} \} \\ & \langle + \cdot \mathsf{s} \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{tl}) \rangle \;=\; \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{tl}) \rangle \\ \equiv & \quad \{ \text{ comutatividade da adição aritmética } (\textit{i.e.}, + \cdot \mathsf{s} = +) \} \\ & \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{tl}) \rangle \;=\; \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{s} \cdot (\mathsf{tl} \times \mathsf{tl}) \rangle \\ \equiv & \quad \{ \text{ igualdade de } \textit{splits} \} \\ & \mathsf{true} \end{array}$$

□

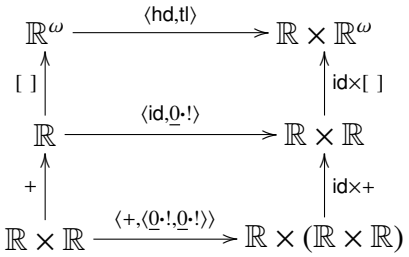
Lema 8

$$[r + p] \;=\; [r] + [p] \tag{3.17}$$

O interesse desta lei (no que se refere às técnicas de prova de cálculo discutidas nesta dissertação) vem do facto de, ao contrário dos casos ditos estruturais, o resultado ser aqui dependente da escolha judiciosa dos argumentos. Por outro lado a tradução da igualdade (3.17) em notação *pointfree* resulta em

$$+ \cdot ([\,] \times [\,]) \;=\; [\,] \cdot + \tag{3.18}$$

onde a lei de fusão-ana não é imediatamente aplicável, pois o segundo membro não está definido como um anamorfismo. Assim a prova será feita em dois momentos. Em primeiro lugar mostraremos que o termo $[\,] \cdot +$ no lado direito é equivalente a um anamorfismo, conforme o diagrama seguinte. Seguidamente aplicaremos a lei da fusão-ana.



Prova. Provemos primeiro que

$$[\,] \cdot + \;=\; [\![\langle +, \langle \underline{0} \cdot !, \underline{0} \cdot ! \rangle \rangle]\!] \tag{3.19}$$

$$\begin{array}{ll}
 & [\,] \cdot + \;=\; [\![\langle +, \langle \underline{0} \cdot !, \underline{0} \cdot ! \rangle \rangle]\!] \\
\equiv & \quad \{ \text{ definição (3.9) } \} \\
 & [\![\langle \mathsf{id}, \underline{0} \cdot ! \rangle]\!] \cdot + \;=\; [\![\langle +, \langle \underline{0} \cdot !, \underline{0} \cdot ! \rangle \rangle]\!] \\
\Leftarrow & \quad \{ \text{ lei fusão-ana } \} \\
 & \langle \mathsf{id}, \underline{0} \cdot ! \rangle \cdot + \;=\; (\mathsf{id} \times +) \cdot \langle +, \langle \underline{0} \cdot !, \underline{0} \cdot ! \rangle \rangle \\
\equiv & \quad \{ \text{ leis fusão-produto e absorção-produto } \} \\
 & \langle +, \underline{0} \cdot ! \cdot + \rangle \;=\; \langle +, + \cdot \langle \underline{0}, \underline{0} \rangle \cdot ! \rangle \\
\equiv & \quad \{ \text{ aritmética } \} \\
 & \langle +, \underline{0} \cdot ! \cdot + \rangle \;=\; \langle +, \underline{0} \cdot ! \rangle \\
\equiv & \quad \{ \text{ !-universal } \} \\
 & \langle +, \underline{0} \cdot ! \cdot + \rangle \;=\; \langle +, \underline{0} \cdot ! \rangle \\
\equiv & \quad \{ \text{ igualdade de } \textit{splits} \} \\
 & \mathsf{true}
\end{array}$$

Mostremos agora o resultado principal

$$
\begin{array}{ll}
 & + \cdot ([\,] \times [\,]) \;=\; [\,] \cdot + \\
\equiv & \quad \{ \text{ definição (3.8) e resultado acabado de provar } \} \\
 & [\!(\langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl} \rangle)\!] \cdot ([\,] \times [\,]) \;=\; [\!(\langle +, \langle \underline{0} \cdot !, \underline{0} \cdot ! \rangle \rangle)\!] \\
\Leftarrow & \quad \{ \text{ lei fusão-ana } \} \\
 & \langle + \cdot (\mathsf{hd} \times \mathsf{hd}), \mathsf{tl} \times \mathsf{tl} \rangle \cdot ([\,] \times [\,]) \;=\; (\mathsf{id} \times ([\,] \times [\,])) \cdot \langle +, \langle \underline{0} \cdot !, \underline{0} \cdot ! \rangle \rangle \\
\equiv & \quad \{ \text{ leis fusão-produto e absorção-produto } \} \\
 & \langle + \cdot (\mathsf{hd} \times \mathsf{hd}) \cdot ([\,] \times [\,]), (\mathsf{tl} \times \mathsf{tl}) \cdot ([\,] \times [\,]) \rangle \;=\; \langle +, ([\,] \times [\,]) \cdot \langle \underline{0} \cdot !, \underline{0} \cdot ! \rangle \rangle \\
\equiv & \quad \{ \text{ functores e lei fusão-produto } \} \\
 & \langle + \cdot (\mathsf{hd} \cdot [\,] \times \mathsf{hd} \cdot [\,]), (\mathsf{tl} \cdot [\,] \times \mathsf{tl} \cdot [\,]) \rangle \;=\; \langle +, ([\,] \cdot \underline{0} \cdot ! \times [\,] \cdot \underline{0} \cdot !) \rangle \\
\equiv & \quad \{ \text{ lei (3.12) } \} \\
 & \langle + \cdot (\mathsf{id} \times \mathsf{id}), [\,] \cdot \underline{0} \cdot ! \times [\,] \cdot \underline{0} \cdot ! \rangle \;=\; \langle +, ([\,] \cdot \underline{0} \cdot ! \times [\,] \cdot \underline{0} \cdot !) \rangle \\
\equiv & \quad \{ \text{ identidades e igualdade de } \textit{slpits} \} \\
 & \mathsf{true}
\end{array}
$$

□

Consideremos agora uma prova alternativa que se baseia no princípio de coindução clássica usada em [Rut00, Rut01], mas recorrendo explicitamente a resultados provados por cálculo acima. A prova é claramente mais simples.

Prova. A prova baseia-se em mostrar que o resultado da aplicação dos dois observadores canónicos de *streams* (*i.e.*, a correspondente coalgebra final) são iguais. Assim,

$$
\begin{array}{ll}
 & \mathsf{hd}\ [r + s] \\
\equiv & \quad \{ \text{ lei (3.12) } \} \\
 & r + s \\
\equiv & \quad \{ \text{ lei (3.12) } \} \\
 & \mathsf{hd}\ [r] \;+\; \mathsf{hd}\ [s]
\end{array}
$$

$$
\begin{array}{ll}
 & \mathsf{tl}\ [r + s] \\
\equiv & \quad \{ \text{ lei (3.12) } \} \\
 & [0] \\
\equiv & \quad \{ \text{ lei (3.13) do lema 6 } \}
\end{array}
$$

$$
\begin{array}{ll}
& [0] + [0] \\
\equiv & \quad \{ \text{ lei (3.12) } \} \\
& \mathsf{tl}\ [r]\ +\ \mathsf{tl}\ [s]
\end{array}
$$

□

Lema 9

$$\sigma + (\tau + \rho) \ = \ (\sigma + \tau) + \rho \tag{3.20}$$

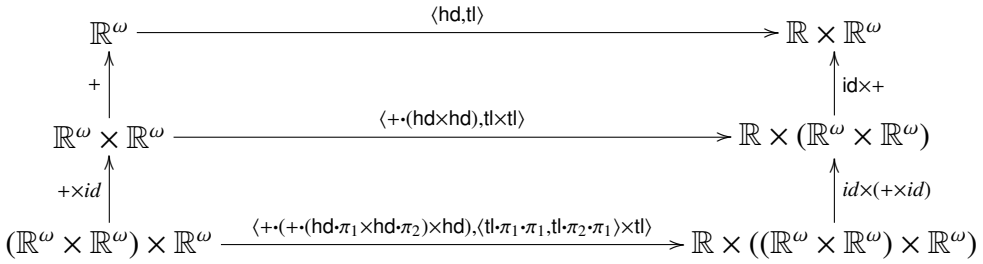
Prova. Em notação *pointfree* a igualdade (3.21) é re-escrita como

$$+ \cdot (id \times +) \cdot a \ = \ + \cdot (+ \times id) \tag{3.21}$$

onde $a = \langle \pi_1 \cdot \pi_1, \langle \pi_2 \cdot \pi_1, \pi_2 \rangle \rangle$ é um isomorfismo natural. Donde

$$
\begin{array}{ll}
& + \cdot (id \times +) \cdot a \ = \ + \cdot (+ \times id) \\
\equiv & \quad \{ \text{ lei absorção-produto } \} \\
& + \cdot \langle id \cdot \pi_1 \cdot \pi_1, + \cdot \langle \pi_2 \cdot \pi_1, \pi_2 \rangle \rangle \ = \ + \cdot (+ \times id)
\end{array}
$$

Para poder usar a lei fusão-ana, tem que se reescrever $+\cdot(+\times id)$ como um anamorfismo. A prova prossegue como esperado.



□

3.3 Apomorfismo

O *apomorfismo* é o funcional que capta a *correcursão primitiva*. A intuição é que funções definidas desse modo são funções cujo gene pode produzir não apenas a continuação imediata do argumento, tal como dada pela dinâmica da coalgebra, mas também um valor específico do tipo coindutivo. Formalmente,

Definição 13 *Seja* $(\nu_{\mathsf{T}}, \mathsf{out}_{\mathsf{T}})$ *uma coalgebra final. Para qualquer função* $\varphi : \mathsf{T}(\mathsf{C}+\nu_{\mathsf{T}}) \longleftarrow \mathsf{C}$*, define-se a função* $\mathsf{apo}\,\varphi : \nu_{\mathsf{T}} \longleftarrow \mathsf{C}$ *como uma composição de um anamorfismo com a injecção esquerda:*

$$\mathsf{apo}\,\varphi = [\!([\varphi, \mathsf{T}(\iota_2) \cdot \mathsf{out}_{\mathsf{T}}])\!] \cdot \iota_1$$

Dito de outro modo,

Corolário 1 *Para toda a função* $\varphi : \mathsf{T}(\mathsf{C} + \nu_{\mathsf{T}}) \longleftarrow \mathsf{C}$*, o apomorfismo* $f = \mathsf{apo}\,\varphi : \nu_{\mathsf{T}} \longleftarrow \mathsf{C}$ *é a única função que faz com que o seguinte diagrama comuta:*

$$\begin{array}{ccc} \mathsf{C} & \xrightarrow{\varphi} & \mathsf{T}(\mathsf{C} + \nu_{\mathsf{T}}) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \mathsf{T}[f,\mathsf{id}]} \\ \nu_{\mathsf{T}} & \xrightarrow{\mathsf{out}_{\mathsf{T}}} & \mathsf{T}\mathsf{out}_{\mathsf{T}} \end{array}$$

ou seja definida pela seguinte propriedade universal

$$\mathsf{out}_{\mathsf{T}} \cdot f = \mathsf{T}[f, \mathsf{id}] \cdot \varphi \equiv f = \mathsf{apo}\,\varphi \tag{3.22}$$

Da propriedade universal (3.22) deduzem-se as seguintes leis, mais uma vez ditas, respectivamente, de *cancelamento*, *reflexão* e *fusão*.

$$\mathsf{out}_{\mathsf{T}} \cdot \mathsf{apo}\,\varphi = \mathsf{T}[\mathsf{apo}\,\varphi, \mathsf{id}] \cdot \varphi \tag{3.23}$$

$$\mathsf{id} = \mathsf{apo}\,\mathsf{T}(\iota_1) \cdot \mathsf{out}_{\mathsf{T}} \tag{3.24}$$

$$\phi \cdot f = \mathsf{T}(f + \mathsf{id}) \cdot \varphi \Rightarrow \mathsf{apo}\,\psi \cdot f = \mathsf{apo}\,\varphi \tag{3.25}$$

onde $\varphi : \mathsf{T}(\mathsf{C}+\nu_{\mathsf{T}}) \longleftarrow \mathsf{C}$, $\psi : \mathsf{T}(\mathsf{C}+\nu_{\mathsf{T}}) \longleftarrow \mathsf{D}$ e $f : \mathsf{D} \longleftarrow \mathsf{C}$. É imediato concluir que todo o anamorfismo $[\!(\varphi)\!]$ é um apomorfismo:

$$\mathsf{apo}\,\mathsf{T}(\iota_1) \cdot \varphi$$

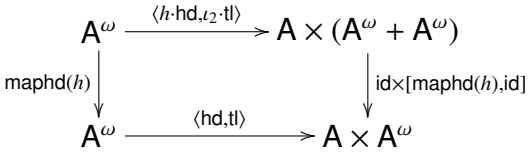
Vejamos, agora, alguns exemplos de definições através de coiteração natural.

1. $\boxed{\mathsf{maphd}(h) : Stream(A) \longleftarrow Stream(A)}$
   consiste em aplicar a função $h : A \longleftarrow A$ à cabeça da lista, enquanto a cauda se mantém inalterável.

   Na perspectiva da coindução explícita de [Rut05] a função define-se por

$$\mathsf{maphd}(h)(\sigma)(0) = h(\sigma(0))$$
$$\mathsf{maphd}(h)' = \sigma'$$

Definição pela Propriedade Universal
Comecemos por representar o diagrama:



Como o diagrama comuta, vem que

$$
\begin{array}{ll}
 & \langle \mathsf{hd}, \mathsf{tl}\rangle \cdot \mathsf{maphd}(h) \;=\; (\mathsf{id} \times [\mathsf{maphd}(h), \mathsf{id}]) \cdot \langle h \cdot \mathsf{hd}, \iota_2 \cdot \mathsf{tl}\rangle \\
\equiv & \quad \{ \text{×-fusão e absorção} \} \\
 & \langle \mathsf{hd} \cdot \mathsf{maphd}(h), \mathsf{tl} \cdot \mathsf{maphd}(h)\rangle \;=\; \langle \mathsf{id} \cdot h \cdot \mathsf{hd}, [\mathsf{maphd}(h), \mathsf{id}] \cdot \iota_2 \cdot \mathsf{tl}\rangle \\
\equiv & \quad \{ \text{pelo cancelamento} \} \\
 & \langle \mathsf{hd} \cdot \mathsf{maphd}(h), \mathsf{tl} \cdot \mathsf{maphd}(h)\rangle \;=\; \langle h \cdot \mathsf{hd}, \mathsf{id} \cdot \mathsf{tl}\rangle \\
\equiv & \quad \{ \text{definição de identidade e pela igualdade de splits} \} \\
 & \mathsf{hd} \cdot \mathsf{maphd}(h) \;=\; h \cdot \mathsf{hd} \\
 & \mathsf{tl} \cdot \mathsf{maphd}(h) \;=\; \mathsf{tl}
\end{array}
$$

Portanto,

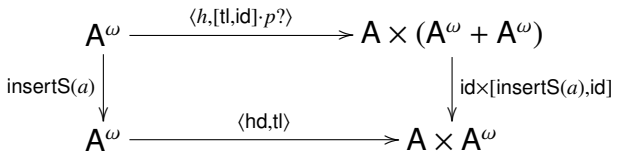
$$\mathsf{maphd}(h) = \mathsf{apo}\,\langle h \cdot \mathsf{hd}, \iota_2 \cdot \mathsf{tl}\rangle$$

2. $\boxed{\mathsf{insertS}(a) : Stream(A) \longleftarrow Stream(A)}$
O objectivo desta função consiste em inserir o elemento $a : A \longleftarrow \mathbf{1}$ de forma ordenada.

De novo, na perspectiva de [Rut05] temos

$$
\begin{array}{rl}
\mathsf{insertS}(a)(\sigma)(0)|a & \text{se } a < (\sigma(0)) \\
|\sigma(0) & \text{caso contrário} \\
\mathsf{insertS}(a)(\sigma)'|(a : \sigma') & \text{se } a < (\sigma(1)) \\
|\mathsf{insertS}(a)(\sigma') & \text{caso contrário}
\end{array}
$$

Definição pela Propriedade Universal
Comecemos por representar o diagrama:



$$\begin{aligned}
&\langle \mathsf{hd},\mathsf{tl}\rangle\cdot\mathsf{insertS}(a) \;=\; (\mathsf{id}\times[\mathsf{insertS}(a),\mathsf{id}])\cdot\langle h,[\mathsf{tl},\mathsf{id}]\cdot p?\rangle\\
\equiv\quad &\{\times\text{-fusão e absorção}\}\\
&\langle \mathsf{hd}\cdot\mathsf{insertS}(a),\mathsf{tl}\cdot\mathsf{insertS}(a)\rangle \;=\; \langle \mathsf{id}\cdot h,[\mathsf{insertS}(a),\mathsf{id}]\cdot[\mathsf{tl},\mathsf{id}]\cdot p?\rangle
\end{aligned}$$

Portanto,

$$\mathsf{insertS(a)} = \mathsf{apo}\,\langle h,[\mathsf{tl},\mathsf{id}]\cdot p?\rangle$$

em que,

$$\begin{array}{lll}
p(xs) & = head(xs)\leq a() & \\
h(xs) & |\,head(xs) & \text{se } p(xs)\\
 & |\,a() & \text{caso contrário}
\end{array}$$

3.3.1 Codificação em HASKELL

Para codificarmos este esquema em HASKELL recorremos aos construtores já definidos na especificação do anamorfismo (ver secção anterior). Notemos, porém, que o tipo do apomorfismo é $\mathsf{C}\to\mathsf{T}(\mathsf{C}+\nu_\mathsf{T})$. Como o HASKELL não tem definido o construtor para a soma, temos que o definir primeiro. Assim,

---

```
data L a x = Empty | C a x

instance Functor (L a)
    where fmap f Empty = Empty
          fmap f (C c x) = C c (f x)

(-|-) :: (a -> b) -> (c -> d) -> Either a c -> Either b d

f -|- g= either (i1 . f) (i2 . g)

class BiFunctor f where
      bmap :: (a -> b) -> (c -> d) -> (f a c -> f b d)
```

```
instance BiFunctor Either where
    bmap f g = f -|- g
```

Depois definimos o apomorfismo:

```
apo :: Functor f => (c -> f (Either c (Nu f))) -> c -> Nu f

apo phi = Fin . fmap (either (apo phi) id) . phi
```

Exemplos de Animação

```
Corecursion> toHList(natS)
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,
58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90{Interrupted!}

Corecursion> toHList(insertS 0 natS)
[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,
21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,
39,40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,
57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,
75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90{Interrupted!}

Corecursion> toHList(maphd (2*) natS)
[2,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,
40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,
58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75,
76,77,78,79,80,81,82,83,84,85,86,87,88,89,90{Interrupted!}
```

3.4 Futumorfismo

Como vimos, o *apomorfismo* contempla situações em que o resultado da função pode ser tanto a continuação imediata como um valor arbitrário do tipo coindutivo. O *futumorfismo*, por seu lado, vai mais longe, disponibilizando como alternativa à continuação imediata, toda uma colecção de valores do tipo coindutivo.

Essa colecção está estruturada num tipo indutivo arbitrário. Esta flexibilidade acrescida torna, por certo, as definições base mais complexas. Em particular, tem que ser tomado em consideração o modo de interacção entre o tipo coindutivo e a estrutura indutiva de suporte. No entanto, é surpreendente como, no estilo de coindução implícita, chegamos, também para este caso, a um conjunto de leis calculacionais simples e fáceis de aplicar.

A definição recorre ao conceito de *cv-coalgebra* para captar a interacção referida. Assim,

Definição 14 *Considere-se o endofunctor* $\mathsf{T} : \mathsf{C} \longleftarrow \mathsf{C}$ *para o qual existe a coalgebra final* $(\nu_\mathsf{T}, \mathsf{out}_\mathsf{T})$. *Define-se o bifunctor* $\mathsf{T}^+ : \mathsf{C} \longleftarrow \mathsf{C} \times \mathsf{C}$ *tal que:*

$$\mathsf{T}^+(A, X) = A + \mathsf{T}(X).$$

Assume que para qualquer objecto A existe a álgebra inicial T^+_A*, ou seja, o par* $(\mu_{\mathsf{T}^+_A}, \mathsf{in}_\mathsf{T})$*, isto é,*$\mathsf{T}^+$ *induz um functor* $\mathsf{T}^\mu(X) = \mu\mathsf{T}^+_X$. *A* T-cv-coalgebra *é um par* (C, φ)*, onde* C *é um objecto e* $\varphi : \mathsf{T}(\mathsf{T}^+(C)) \longleftarrow C$ *é uma função.*

Definição 15 *Sejam* (C, φ) *e* (D, ψ) *duas* T-cv-coalgebras. *Um* homomorfismo *de* (C, φ) *para* (D, ψ) *é uma função* $f : D \longleftarrow C$ *na categoria* C*, tal que*

$$\psi \cdot f = \mathsf{T}(\!|\mathsf{in}_\mathsf{T} \cdot (f + \mathsf{id})|\!).\varphi$$

o que faz comutar o seguinte diagrama

$$\begin{array}{ccc} C & \xrightarrow{\varphi} & \mathsf{T}(\mathsf{T}^+(C)) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \mathsf{T}(\!|\mathsf{in}_\mathsf{T}\cdot(f+\mathsf{id})|\!)} \\ D & \xrightarrow{\psi} & \mathsf{T}(\mathsf{T}^\mu(D)) \end{array}$$

Podemos, agora, definir o *futumorfismo* por uma propriedade universal,

Definição 16 *Seja* $(\nu_\mathsf{T}, \mathsf{out}_\mathsf{T})$ *uma coalgebra final. Para qualquer* T*-cv-coalgebra* $\varphi : \mathsf{T}(\mathsf{T}^+(C)) \longleftarrow C$*, a função* $\mathsf{futu}\varphi : \nu_\mathsf{T} \longleftarrow C$ *é definida por*

$$\mathsf{futu}\varphi = [\!([\varphi, \mathsf{id}] \cdot \mathsf{in}_\mathsf{T}^{-1})]\!] \cdot \mathsf{in}_\mathsf{T} \cdot \iota_1$$

A função $\mathsf{futu}\varphi$ *é chamada de futumorfismo.*

Corolário 2 *Para qualquer* T*-cv-coalgebra* $\varphi : \mathsf{T}(\mathsf{T}^+(C)) \longleftarrow C$*, o futumorfismo* $f = \mathsf{futu}\varphi : \nu_\mathsf{T} \longleftarrow C$ *é a única função que faz com que o seguinte diagrama comute:*

$$\begin{array}{ccc} C & \xrightarrow{\varphi} & \mathsf{T}(\mathsf{T}^+(C)) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle \mathsf{T}(\!|[f,\mathsf{out}_\mathsf{T}^{-1}]|\!)} \\ \nu_\mathsf{T} & \xrightarrow{\mathsf{out}_\mathsf{T}} & \mathsf{T}(\nu_\mathsf{T}) \end{array}$$

o que corresponde à seguinte propriedade universal:

$$\mathsf{out_T} \cdot f = \mathsf{T}(\!|[f, \mathsf{out_T^{-1}}]|\!) \cdot \varphi \equiv f = \mathsf{futu}\varphi$$

As seguintes leis[3] são deduzidas da propriedade universal, considerando $(\mu_\mathsf{T}, \mathsf{in_T})$. São elas, respectivamente, as leis de *cancelamento*, *reflexão*, *fusão* e a que reduz um anamorfismo a um futumorfismo.

$$\mathsf{out_T} \cdot \mathsf{futu}\varphi = \mathsf{T}\,(\!|[\mathsf{futu}\varphi, \mathsf{out_T^{-1}}]|\!) \cdot \varphi \tag{3.26}$$

$$\mathsf{futuT}\,(\mathsf{in_T} \cdot \iota_1) \cdot \nu_\mathsf{T} = \mathsf{id}\nu_\mathsf{T} \tag{3.27}$$

$$\mathsf{futu}\psi \cdot h = \mathsf{futu}\varphi \quad \text{se} \quad \psi \cdot h = \mathsf{T}\,(\!|[\mathsf{in_T} \cdot (h + \mathsf{id})]|\!) \cdot \varphi \tag{3.28}$$

$$[\!(\varphi)\!] = \mathsf{futuT}\,(\mathsf{in_T} \cdot \iota_1) \cdot \varphi \tag{3.29}$$

Terminaremos esta subsecção com a discussão de uma função expressa como um *futumorfismo* e a verificação de algumas propriedades. Trata-se da função

$$\boxed{\mathsf{exch} : Stream(A) \longleftarrow Stream(A)}$$

cujo objectivo é realçar a permuta dos elementos da stream argumento dois a dois.

Em [Rut05] esta função é definida como

$$\begin{aligned} \mathsf{exch}(\sigma)(0) &= \sigma(1) \\ \mathsf{exch}(\sigma)' &= \sigma(0) : \mathsf{exch}(\sigma'') \end{aligned}$$

o que nos permite escrever, na notação dessa referência,

$$\mathsf{exch}(\sigma) = (\sigma(1), \sigma(0), \sigma(2), \sigma(1), \sigma(3), \sigma(2), \ldots)$$

Vejamos, agora, como se define a mesma função pela propriedade universal identificada acima. Comecemos por representar o diagrama:

$$\begin{array}{ccc} \mathsf{A}^\omega & \xrightarrow{\langle \mathsf{hd}\cdot\mathsf{tl}, \mathsf{cons}\cdot\langle \mathsf{hd}, \mathsf{sing}\cdot\mathsf{tl}\rangle\rangle} & \mathsf{A} \times \mathsf{A}\mathsf{A}^{\omega+} \\ {\scriptstyle \mathsf{exch}}\downarrow & & \downarrow{\scriptstyle \mathsf{id}\times(\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!)} \\ \mathsf{A}^\omega & \xrightarrow{\langle \mathsf{hd}, \mathsf{tl}\rangle} & \mathsf{A} \times \mathsf{A}^\omega \end{array}$$

[3] Tal como nos outros casos, o leitor é referido a [Ven00] para a sua demonstração cuidada.

Por sua vez o *catamorfismo* auxiliar[4]

$$\begin{array}{ccc} A^{\omega} + A \times AA^{\omega+} & \xrightarrow{[sing,cons]} & AA^{\omega+} \\ {\scriptstyle id+id\times(\!|[exch,cons^{-1}]|\!)}\downarrow & & \downarrow{\scriptstyle (\!|[exch,cons^{-1}]|\!)} \\ A^{\omega} + A \times A^{\omega} & \xrightarrow{[exch,cons^{-1}]} & A^{\omega} \end{array}$$

Como este diagrama comuta, temos que

$$[exch, cons^{-1}] \cdot (id + id \times (\!|[exch, cons^{-1}]|\!)) \;=\; (\!|[exch, cons^{-1}]|\!) \cdot [sing, cons]$$

Concentremo-nos, agora, no diagrama do futumorfismo. Pretendemos provar que este comuta, ou seja,

$$\begin{array}{ll} & \langle hd, tl\rangle \cdot exch \;=\; (id \times [exch, cons^{-1}]) \cdot \langle hd \cdot tl, cons \cdot \langle hd, sing \cdot tl\rangle\rangle \\ \equiv & \quad \{\text{ $\times$-fusão e absorção}\} \\ & \langle hd \cdot exch, tl \cdot exch\rangle \;=\; \langle id \cdot hd \cdot tl, (\!|[exch, cons^{-1}]|\!) \cdot cons \cdot \langle hd, sing \cdot tl\rangle\rangle \\ \equiv & \quad \{\text{ definição de identidade e pela igualdade de splits}\} \\ & hd \cdot exch \;=\; hd \cdot tl \\ & tl \cdot exch \;=\; (\!|[exch, cons^{-1}]|\!) \cdot cons \cdot \langle hd, sing \cdot tl\rangle \end{array}$$

A primeira igualdade vem directamente da definição da função exch. A segunda, porém, terá que ser provada. Notemos que a expressão $(\!|[exch, cons^{-1}]|\!) \cdot cons \cdot \langle hd, sing \cdot tl\rangle$ não se pode simplificar directamente. Façamos, então, uma generalização e consideremos a expressão completa, *i.e.*, tomemos a composição com a álgebra inicial [sing, cons] e não apenas o construtor cons. Assim,

$$(\!|[exch, cons^{-1}]|\!) \cdot [sing, cons] \cdot id + \langle hd, sing \cdot tl\rangle$$

Vamos agora aplicar as leis do catamorfismo [BM97] a essa expressão:

[4] O *catamorfismo* é o dual do anamorfismo, representando o único morfismo, na categoria das álgebras de um dado functor, com origem na álgebra inicial e destino numa álgebra especificada. É, assim, a versão *calculacional*, ou *implícita*, da indução estrutural.

$$
\begin{aligned}
& (\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!) \cdot [\mathsf{sing}, \mathsf{cons}] \cdot \mathsf{id} + \langle \mathsf{hd}, \mathsf{sing} \cdot \mathsf{tl} \rangle \\
\equiv & \quad \{ \text{pelo cancelamento cata} \} \\
& [\mathsf{exch}, \mathsf{cons}^{-1}] \cdot (\mathsf{id} + \mathsf{id} \times (\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!)) \cdot (\mathsf{id} + \langle \mathsf{hd}, \mathsf{sing} \cdot \mathsf{tl} \rangle) \\
\equiv & \quad \{ \text{pela absorção - +} \} \\
& [\mathsf{exch}, \mathsf{cons}^{-1} \cdot (\mathsf{id} \times (\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!)) \cdot \langle \mathsf{hd}, \mathsf{sing} \cdot \mathsf{tl} \rangle] \\
\equiv & \quad \{ \text{pela absorção - } \times \} \\
& [\mathsf{exch}, \mathsf{cons}^{-1} \cdot \langle \mathsf{hd}, (\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!) \cdot \mathsf{sing} \cdot \mathsf{tl} \rangle]
\end{aligned}
$$

Por outro lado,

$$
\begin{aligned}
& (\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!) \cdot [\mathsf{sing}, \mathsf{cons}] \cdot \mathsf{id} + \langle \mathsf{hd}, \mathsf{sing} \cdot \mathsf{tl} \rangle \\
\equiv & \quad \{ \text{pela fusão - +} \} \\
& [(\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!) \cdot \mathsf{sing}, (\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!) \cdot \mathsf{cons}] \cdot (\mathsf{id} + \langle \mathsf{hd}, \mathsf{sing} \cdot \mathsf{tl} \rangle) \\
\equiv & \quad \{ \text{pela absorção - +} \} \\
& [(\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!) \cdot \mathsf{sing}, (\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!) \cdot \mathsf{cons} \cdot \langle \mathsf{hd}, \mathsf{sing} \cdot \mathsf{tl} \rangle]
\end{aligned}
$$

Igualando as expressões obtidas, vem, pela igualdade do either,

$$
\begin{aligned}
\mathsf{exch} &= (\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!) \cdot \mathsf{sing} \\
\mathsf{cons}^{-1} \cdot \langle \mathsf{hd}, (\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!) \cdot \mathsf{sing} \cdot \mathsf{tl} \rangle &= (\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!) \cdot \mathsf{cons} \cdot \langle \mathsf{hd}, \mathsf{sing} \cdot \mathsf{tl} \rangle
\end{aligned}
$$

Substituindo $(\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!) \cdot \mathsf{sing}$ na segunda expressão concluímos que

$$\mathsf{cons}^{-1} \cdot \langle \mathsf{hd}, \mathsf{exch} \cdot \mathsf{tl} \rangle = (\!|[\mathsf{exch}, \mathsf{cons}^{-1}]|\!) \cdot \mathsf{cons} \cdot \langle \mathsf{hd}, \mathsf{sing} \cdot \mathsf{tl} \rangle$$

Da mesma forma podemos concluir que

$$\mathsf{tl} \cdot \mathsf{exch} = \mathsf{cons}^{-1} \cdot \langle \mathsf{hd}, \mathsf{exch} \cdot \mathsf{tl} \rangle$$

ou seja,

$$
\begin{aligned}
\mathsf{hd} \cdot \mathsf{tl} \cdot \mathsf{exch} &= \mathsf{hd} \\
\mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{exch} &= \mathsf{exch} \cdot \mathsf{tl}
\end{aligned}
$$

Assim se conclui que a função exch pode ser vista como um futumorfismo[5], *i.e.*

[5]Chama-se a atenção para uma gralha detectada em [Ven00], página 58, onde aparece, na definição de uma função semelhante, uma aplicação extra do observador tl que é indevida.

$$\mathsf{exch} = \mathsf{futu}\langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{cons} \cdot \langle \mathsf{hd}, \mathsf{sing} \cdot \mathsf{tl} \rangle \rangle$$

Por observação do exemplo apresentado, concluímos que as posições pares da aplicação da função exch a uma stream correspondem à sua cauda e as ímpares à própria stream. Assim, formulamos a seguinte propriedade:

$$\begin{aligned} \mathsf{even} \cdot \mathsf{exch} &= \mathsf{tl} \\ \mathsf{odd} \cdot \mathsf{exch} &= \mathsf{id} \end{aligned}$$

Prova. Como a função even e a função odd podem ser escritas como anamorfismos, convém-nos escrever as funções tl e id também como anamorfismos, para podermos recorrer à lei de fusão-ana. De forma trivial, podemos escrever

$$\begin{aligned} \mathsf{tl} &= [\!(\langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \rangle)\!] \\ \mathsf{id} &= [\!(\langle \mathsf{hd}, \mathsf{tl} \rangle)\!] \end{aligned}$$

Observemos os respectivos diagramas.

$$\begin{array}{ccc} \mathsf{A}^\omega & \xrightarrow{\mathsf{tl}} & \mathsf{A}^\omega \\ {\scriptstyle \langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \rangle} \downarrow & & \downarrow {\scriptstyle \langle \mathsf{hd}, \mathsf{tl} \rangle} \\ \mathsf{A} \times \mathsf{A}^\omega & \xrightarrow{\mathsf{id} \times \mathsf{tl}} & \mathsf{A} \times \mathsf{A}^\omega \end{array}$$

$$\begin{aligned} & \langle \mathsf{hd}, \mathsf{tl} \rangle \cdot \mathsf{tl} = (\mathsf{id} \times \mathsf{tl}) \cdot \langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \rangle \\ \equiv & \quad \{ \text{por fusão e absorção} \} \\ & \langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \cdot \mathsf{tl} \rangle = \langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \cdot \mathsf{tl} \rangle \end{aligned}$$

$$\begin{array}{ccc} \mathsf{A}^\omega & \xrightarrow{\mathsf{id}} & \mathsf{A}^\omega \\ {\scriptstyle \langle \mathsf{hd}, \mathsf{tl} \rangle} \downarrow & & \downarrow {\scriptstyle \langle \mathsf{hd}, \mathsf{tl} \rangle} \\ \mathsf{A} \times \mathsf{A}^\omega & \xrightarrow{\mathsf{id} \times \mathsf{id}} & \mathsf{A} \times \mathsf{A}^\omega \end{array}$$

$$\begin{aligned} & \langle \mathsf{hd}, \mathsf{tl} \rangle \cdot \mathsf{id} \;=\; (\mathsf{id} \times \mathsf{id}) \cdot \langle \mathsf{hd}, \mathsf{tl} \rangle \\ \equiv & \quad \{ \text{por fusão e absorção} \} \\ & \langle \mathsf{hd}, \mathsf{tl} \rangle \;=\; \langle \mathsf{hd}, \mathsf{tl} \rangle \end{aligned}$$

Passemos agora às provas pretendidas.

- $\mathsf{even} \cdot \mathsf{exch} = \mathsf{tl}$
  i.e.

$$
\begin{array}{ll}
 & [\![(\langle \mathsf{hd}, \mathsf{tl} \cdot \mathsf{tl} \rangle)]\!] \cdot \mathsf{exch} \;=\; [\![(\langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \rangle)]\!] \\
\Leftarrow & \quad \{ \text{pela lei de fusão-ana} \} \\
 & \langle \mathsf{hd}, \mathsf{tl} \cdot \mathsf{tl} \rangle \cdot \mathsf{exch} \;=\; (\mathsf{id} \times \mathsf{exch}) \cdot \langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \rangle \\
\equiv & \quad \{ \text{pela fusão-}\times \text{ e absorção - }\times \} \\
 & \langle \mathsf{hd} \cdot \mathsf{exch}, \mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{exch} \rangle \;=\; \langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{exch} \cdot \mathsf{tl} \rangle \\
\equiv & \quad \{ \text{pela igualdade de splits} \} \\
 & \mathsf{hd} \cdot \mathsf{exch} \;=\; \mathsf{hd} \cdot \mathsf{tl} \\
 & \mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{exch} \;=\; \mathsf{exch} \cdot \mathsf{tl}
\end{array}
$$

- $\mathsf{odd} \cdot \mathsf{exch} = \mathsf{id}$
  i.e.

$$
\begin{array}{ll}
 & [\![(\langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \cdot \mathsf{tl} \rangle)]\!] \cdot \mathsf{exch} \;=\; [\![(\langle \mathsf{hd}, \mathsf{tl} \rangle)]\!] \\
\Leftarrow & \quad \{ \text{pela lei de fusão-ana} \} \\
 & \langle \mathsf{hd} \cdot \mathsf{tl}, \mathsf{tl} \cdot \mathsf{tl} \rangle \cdot \mathsf{exch} \;=\; (\mathsf{id} \times \mathsf{exch}) \cdot \langle \mathsf{hd}, \mathsf{tl} \rangle \\
\equiv & \quad \{ \text{pela fusão-}\times \text{ e absorção - }\times \} \\
 & \langle \mathsf{hd} \cdot \mathsf{tl} \cdot \mathsf{exch}, \mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{exch} \rangle \;=\; \langle \mathsf{hd}, \mathsf{exch} \cdot \mathsf{tl} \rangle \\
\equiv & \quad \{ \text{pela igualdade de splits} \} \\
 & \mathsf{hd} \cdot \mathsf{tl} \cdot \mathsf{exch} \;=\; \mathsf{hd} \\
 & \mathsf{tl} \cdot \mathsf{tl} \cdot \mathsf{exch} \;=\; \mathsf{exch} \cdot \mathsf{tl}
\end{array}
$$

□

3.4.1 Codificação em HASKELL

Para definirmos o futumorfismo, que trabalha com $C \rightarrow \mathsf{T}(\mathsf{T}^{\mu}(C))$, temos que definir o functor de base do tipo de dados $\mathsf{T}^{\mu}(C)$.

```
newtype EitherF f a x = EitherF (Either a (f x))

instance Functor f => Functor (EitherF f a) where
    fmap f (EitherF (Left a))  =  EitherF (Left a)
    fmap f (EitherF (Right x)) =  EitherF (Right (fmap f x))

lastF :: c -> Mu (EitherF f c)
lastF x = In (EitherF (i1 x))
```

```
consF :: f (Mu (EitherF f c)) -> Mu (EitherF f c)
consF x = In (EitherF (i2 x))

joinF :: (a -> c) -> (f b -> c) -> EitherF f a b -> c
joinF f g (EitherF x) = join f g x
```

Agora estamos em condições para definir o futumorfismo:

```
futu :: Functor f => (c -> f (Mu (EitherF f c))) -> c -> Nu f
futu phi = ana (joinF phi id . unIn) . lastF
```

Para efeitos de ilustração, vejamos o seguinte exemplo:

```
Corecursion> toHList(exch natS)
[2,1,3,2,4,3,5,4,6,5,7,6,8,7,9,8,10,9,11,10,12,11,13,12,
14,13,15,14,16,15,17,16,18,17,19,18,20,19,21,20,22,21,23,
22,24,23,25,24,26,25,27,26,28,27,29,28,30,29,31,30,32,31,
33,32,34,33,35,34,36,35,37,36,38,37,39,38,40,39,41,40,42,
41,43,42,44,43,45,44,46{Interrupted!}
```

Capítulo 4

Estudo de Caso: Álgebra de Processos

Resumo

Em [Bar01b] e [BO02] foi estudada de forma sistemática a especificação coindutiva e pointfree *de uma família de álgebras de processos clássicas. Este capítulo reporta a continuação desse trabalho, com ênfase particular na definição de combinadores através de apomorfismos, no estudo do tipo de provas associadas e na construção de um animador de álgebras de processos em* HASKELL*, enquanto linguagem funcional* não estrita*. Numa segunda parte do capítulo é explorado o recurso ao cálculo relacional como ferramenta alternativa de raciocínio. Os dois níveis, coalgébrico e relacional, são relacionados por uma transposição canónica. Esta incursão revela-se muito útil para abordar a formulação coalgébrica da bissimulação fraca, tópico que constitui outra das contribuições do capítulo.*

4.1 Introdução

Em Ciências da Computação designa-se por *álgebra de processos* um conjunto de formalismos desenvolvidos a partir do trabalho pioneiro de Robin Milner, em CCS [Mil80, Mil89], e C. A. R. Hoare, em CSP [Hoa85], para especificar e raciocinar sobre sistemas computacionais exibindo comportamentos complexos e interactivos, com elevado grau de concorrência e distribuição. Sumariamente, diremos que um sistema exibe, ou realiza, comportamento concorrente sempre que lida *simultaneamente* com diferentes *estímulos*, agregando componentes *autónomas* que *interagem ao longo* (e, não apenas, no ínicio e no fim) da sua execução. Adicionalmente o comportamento será *distribuído* se a informação e a capacidade de

processamento (*i.e.*, as entidades e os processos, ou, ainda, os dados e as operações) estiverem disseminados pelas componentes do sistema.

Nesta área incluem-se sistemas tão diversos como gestores de redes de serviços (bancários, administrativos, etc.), sistemas operativos distribuídos, controladores de transportes, telefones ou instrumentação industrial, componentes de interacção homem-máquina, gestores distribuídos da produção, entre muitos outros. Em todos os casos a descrição deste tipo de sistemas enfatiza os aspectos *comportamentais*, i.é, os padrões de ocorrência dos acontecimentos e a sua evolução no tempo e as suas interacções. De facto, enquanto os sistemas de informação tradicionais focam-se na estrutura da informação manipulada e suas transformações, estes sistemas preocupam-se, sobretudo, com o elemento dual, *i.e.*a observação dos comportamentos das suas componentes e suas interacções. Daí serem facilmente descritos em termos de sistemas de transição e, portanto, formalizáveis coindutivamente.

Sistemas de transição de estados, de resto, podem ser encontrados em diversos locais do quotidiano, tendo aplicações simples tais como em relógios despertadores, nas caixas multibanco, em máquinas de selecção de produtos (tais como máquinas de bebidas em que se introduz uma moeda, selecciona-se o produto e a máquina devolve o produto). Por exemplo, nas caixas multibanco introduz-se o cartão, digita-se o código e passa-se à selecção da operação pretendida. Há uns tempos atrás, depois de executar a operação a máquina devolvia o cartão, hoje, há a possibilidade de voltar a digitar o código e seleccionar outra operação. Todos estes sistemas funcionam como 'caixas negras': os seus estados internos podem ou não ser distinguidos apenas pelo *comportamento observado*. Quando o comportamento observado é semelhante, dizemos que os processos são *comportamentalmente equivalentes*. Tal equivalência corresponde exactamente à que é captada pela existência de uma *bissimulação* que os relacione.

Os conceitos base são aqui os de *acção* e *processo*. O primeiro diz respeito a um *evento* que ocorre autonomamente de forma atómica (*i.e.*, indivisível no tempo), funcionando como um estímulo, ponto (porta) de comunicação capaz de ser observado (por outras componentes do sistema ou pelo seu ambiente). O segundo designa um *padrão* de evolução das capacidades de interacção de um sistema ao longo do tempo, *i.e.*, o seu *comportamento*. Assim, uma álgebra de processos fornece uma linguagem (uma teoria e um cálculo) capaz de exprimir interações complexas, os eventos em que, num dado instante, um processo se pode comprometer e os comportamentos que resultam da sua eventual efectivação.

Exemplo. Ilustremos brevemente estas ideias com um exemplo descrito numa das

primeiras, e mais populares, álgebras de processos: o *Calculus of Communicating Systems* (CCS), introduzido por Robin Milner. Neste formalismo a expressão

$$B \triangleq in.\overline{out}.B$$

representa o processo que gere uma célula de memória, por exemplo num computador, acessível por duas acções designadas por *in* (que sinaliza a entrada de informação) e $\overline{out}$ (que sinaliza a saída correspondente). O comportamento da célula resume-se à oferta alternada destas duas possibilidades de interação.

O CCS assume uma *disciplina de interacção* segunda a qual dois processos compostos *concorrencialmente* (através do operador |) *interagem* quando realizam simultaneamente um par de acções *complementares*. Acções complementares representam-se por símbolos idênticos a menos de uma barra horizontal escrita por cima, por exemplo a e $\overline{a}$. O resultado de qualquer sincronização é a acção não observável representada pela letra τ. Assim, podemos compor duas cópias do processo B para construir uma memória de duas posições, ligando a porta $\overline{out}$ do primeiro à porta *in* do segundo. Definimos primeiro cópias de B, renomeando as portas de forma adequada, por recurso ao mecanismo usual de substituição de variáveis num termo. O seu objectivo é assegurar que a acção $\overline{out}$ do primeiro *buffer* se torne complementar da acção *in* do segundo, estabelecendo uma possibilidade de comunicação. As duas componentes são, então, compostas concorrencialmente e a interacção, através da porta m, é tornada interna, por *restricção* (operador $\backslash_C$). Esta internalização significa que as interacções m e $\overline{m}$ deixam de estar disponíveis para o exterior. Assim,

$$\begin{aligned} B_1 &\triangleq B[\overline{m}(x)/\overline{out}(x)] = in.\overline{m}(x).B_1 \\ B_2 &\triangleq B[m(x)/in(x)] = m.\overline{out}(x).B_2 \\ S &\triangleq (B_1 \mid B_2)\backslash_{\{m\}} \end{aligned}$$

✓

Como facilmente se compreende, as álgebras de processos têm vindo a ser uma área de aplicação típica da teoria das coalgebras e do raciocínio coindutivo — *cf.*, por exemplo, [Sch98, Wol99, Bal00, Bar01b]. Neste capítulo iremos prosseguir, como segundo estudo de caso em programação coindutiva, o trabalho reportado nesta última referência (e também em [BO02]), onde é proposta uma reconstrução coindutiva das álgebras de processos clássicas, *i.e.*, sem *mobilidade*. Esse trabalho é sumariamente revisto na secção seguinte. As contribuições específicas deste capítulo são as seguintes:

- Construção de um *animador* e *visualizador* para álgebras de processos em HASKELL (secção 4.3).

- Investigação da aplicação dos princípios de definição e prova por coindução ao desenvolvimento de álgebras de processos. Em particular, na secção 4.4 são introduzidos operadores não *standard* como apomorfismos e investigada a prova de algumas das suas propriedades. A secção introduz ainda uma generalização ao caso dos apomorfismos da lei de fusão condicional proposta em [Bar01b].

- Exploração, na secção 4.5, do cálculo relacional [BH93, BM97] no raciocínio por cálculo em álgebras de processos, a partir da transposição de coalgebras para o functor $\mathcal{P}(Act \times \mathsf{Id})$ para relações binárias. Este estudo permite, de seguida, efectuar, na secção 4.6, a

- caracterização coalgébrica de uma noção de *bissimulação fraca* em que se baseiam as *equivalências observacionais* típicas em álgebra de processos mas cujo tratamento coalgébrico é ainda insuficiente.

4.2 Especificação Coindutiva de Álgebras de Processos

A aproximação coindutiva às álgebras de processos proposta em [Bar01b, BO02] visa aplicar técnicas coalgébricas e o estilo de raciocínio equacional e *pointfree* subjacente ao formalismo de Bird-Meertens [BM97] a esta área das Ciências da Computação. Nesta abordagem processos são definidos como habitantes do portador da coalgebra final para o funtor $\mathcal{P}(Act \times \mathsf{Id})$, onde Act designa um conjunto de identificadores de acções[1]. Por seu lado, os combinadores de processos são definidos coindutivamente como anamorfismos para essa coalgebra final.

Esta aproximação apresenta as seguintes vantagens quando comparada com as abordagens tradicionais:

- Fornece um tratamento uniforme dos *processos* e outras estruturas computacionais, em particular as *estruturas de dados*, ambas representadas como álgebras ou coalgebras para functors que captam, respectivamente, assinaturas de observadores ou construtores. Como se verá na secção seguinte,

[1] Em rigor, a abordagem em [Bar01b] é paramétrica numa componente do functor, permitindo, nomeadamente, substituir o functor *powerset* por outro monad [Mog91] com determindas características. Não elaboraremos nessa direcção na presente tese.

este facto pode ser aproveitado para animar processos em linguagens de programação funcionais.

- Estabelece as provas de propriedades sobre processos numa base *puramente calculacional*, e essencialmente equacional e *pointfree*, deste modo evitando a construção explícita de bissimulações usada na literatura.

- Torna mais simples generalizar definições e resultados, instanciando a abordagem para diferentes disciplinas de interacção e mesmo modelos de comportamento.

Seja, então

$$\omega : \mathcal{P}(Act \times \nu) \longleftarrow \nu$$

a coalgebra final que constitui, nesta abordagem, o universo semântico dos processos. O seu portador, ν, contém classes de equivalência para equivalência comportamental $\sim$ de árvores de ramificação finita etiquetadas por acções em Act.
Sobre ela definem-se dois conjuntos de combinadores: os combinadores *dinâmicos*, *i.e.*, que são consumidos na ocorrência de acções, e os *estáticos* que são persistentes através das transições de estado. No primeiro grupo inclui-se, tipicamente, a *inação*, o *prefixo* e a *escolha*. No segundo, as diferentes formas de *composição paralela*, a *restrição* e a *renomeação* de portas. Recordemos, de [Bar01b] as definições destes combinadores. O leitor é remetido a essa referência para o estudo das suas propriedades.

Combinadores Dinâmicos

Os combinadores neste grupo, sendo não recursivos, são definidos coindutivamente de forma directa. Isto significa que é suficiente especificar, para cada um deles, as observações que origina sob a dinâmica da colagebra final ω. Assim,

- **Inacção**
  O processo inactivo é representado pela constante $\mathsf{nil} : \nu \longleftarrow 1$ e constitui um processo incapaz de qualquer tipo de interacção. É definido coindutivamente como[2]

$$\omega \cdot \mathsf{nil} = \underline{\emptyset}$$

- **Prefixação**
  A prefixação diz respeito ao modo mais usual de se introduzir uma extensão

[2]Note-se que para um valor qualquer $v \in V$, a notação $\underline{v}$ representa a função constante $\underline{v} : V \longleftarrow \mathbf{1}$.

no cálculo. Trata-se de uma família de operadores $a. : \nu \longleftarrow \nu$, indexados por acções $a \in Act$. O significado intuitivo de um termo $a.P$ é o de um processo que realiza a e depois se comporta como P. Formalmente,

$$\omega \cdot a. = \mathsf{sing} \cdot \mathsf{label}_a$$

onde $\mathsf{label}_a = \langle \underline{a}, \mathsf{id} \rangle$ e sing é a função que constroi um conjunto singular com o seu argumento, *i.e.*, $\mathsf{sing}\, v = \{v\}$.

- **Escolha**
  A escolha, $+ : \nu \longleftarrow \nu$, como o próprio nome indica, representa um processo que se comporta como um ou outro dos seus argumentos, escolha essa que é feita de forma não determinística.

  $$\omega \cdot + = \cup \cdot (\omega \times \omega)$$

  Portanto, o processo $P + Q$ comporta-se como P ou como Q.

Combinadores Estáticos

A persistência através da ocorrência de acções força a definição destes combinadores de modo recursivo, *i.e.*, através de um anamorfismo.

- **Entrelaçamento**
  Este operador coloca os processos argumentos em paralelo mas não permitindo qualquer interacção entre eles. As observações do processo composto são exactamente os entrelaçamentos (ou intercalações) das observações de cada um dos processos argumentos. Formalmente,

  $$||| \;=\; [\![(\alpha_{|||})]\!]$$

  onde

  $$\begin{aligned} \alpha_{|||} \;=\; & \nu \times \nu \xrightarrow{\Delta} (\nu \times \nu) \times (\nu \times \nu) \\ & \xrightarrow{(\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)} (\mathcal{P}(Act \times \nu) \times \nu) \times (\nu \times \mathcal{P}(Act \times \nu)) \\ & \xrightarrow{\tau_r \times \tau_l} \mathcal{P}(Act \times (\nu \times \nu)) \times \mathcal{P}(Act \times (\nu \times \nu)) \xrightarrow{\cup} \mathcal{P}(Act \times (\nu \times \nu)) \end{aligned}$$

  Recorde-se que

  $$\begin{aligned} \tau_r &: \mathcal{P}(Act \times (\nu \times \nu)) \longleftarrow \mathcal{P}(Act \times \nu) \times \nu \\ \tau_l &: \mathcal{P}(Act \times (\nu \times \nu)) \longleftarrow \nu \times \mathcal{P}(Act \times \nu) \end{aligned}$$

são as transformações naturais referidas no capítulo 2 que permitem internalizar no functor $\mathcal{P}(Act \times \mathsf{Id})$ informação de contexto[3].

- **Restrição**
  Dado um subconjunto $K \subseteq L$, o combinador restrição $\backslash_K$ impossibilita a ocorrência de quaisquer acções em K. Formalmente,

  $$\backslash_K = [\![(\alpha_{\backslash_K})]\!]$$

  onde

  $$\alpha_{\backslash_K} = \nu \xrightarrow{\omega} \mathcal{P}(Act \times \nu) \xrightarrow{\mathsf{filtro_K}} \mathcal{P}(Act \times \nu)$$

  onde $\mathsf{filtro_K} = \lambda s.\{t \in s \mid \pi_1 t \notin K\}$.

Os combinadores seguintes lidam não apenas com a estrutura das observações, captada pela definição do functor $\mathcal{P}(Act \times \mathsf{Id})$, mas também com a disciplina de interacção entre as acções. Referimos já, no exemplo da secção anterior, que em CCS essa disciplina se resume à possibilidade de sincronização de acções complementares. No entanto essa não é a única possibilidade. De modo a proceder com alguma generalidade, considera-se que a especificação de uma álgebra de processos inclui sempre a definição de uma *álgebra de interacção* sobre Act. Trata-se de um monóide positivo e abeliano $\langle Act; \theta, 1\rangle$ com elemento zero 0. A intuição é de que a operação θ determina a disciplina de interação, enquanto 0 representa a ausência de interacção: *i.e.*, para todo o $a \in Act$, $a\ \theta\ 0 = 0$. Por outro lado, o facto de o monóide ser positivo significa que $a\ \theta\ a' = 1$ sse $a = a' = 1$ [4]. Na secção seguinte ilustramos a codificação em HASKELL de diversas álgebras de interacção. Retomemos, por agora, a especificação dos combinadores estáticos.

- **Renomeação**
  Uma vez fixada uma estrutura de interação, qualquer homomorfismo $f : Act \longleftarrow Act$ pode ser estendido a um operador de renomeação $[f]$ entre processos:

  $$[f] = [\![(\alpha_{[f]})]\!]$$

  onde

  $$\alpha_{[f]} = \nu \xrightarrow{\omega} \mathcal{P}(Act \times \nu) \xrightarrow{\mathcal{P}(f \times \mathsf{id})} \mathcal{P}(Act \times \nu)$$

[3] Tecnicamente, tratam-se das *right* e *left strength* associadas ao monad respectivo.

[4] Em alguns casos é conveniente tomar 1 como representando uma acção sem capacidade para alterar o estado. No entanto a motivação para a sua consideração é essencialmente técnica, de forma a dotar o functor com a estrutura de um monad, o que não seria o caso se tomássemos apenas um semigrupo abeliano (ver [Bar01b] para detalhes).

- **Produto**
  O produto síncrono corresponde à execução simultânea de dois processos. Em cada passo, a acção resultante é determinada pela álgebra de interacção do cálculo. Formalmente,

$$\otimes \;=\; [\![(\alpha_{\otimes})]\!]$$

  onde

$$\alpha_{\otimes} \;=\; \nu \times \nu \xrightarrow{(\omega\times\omega)} \mathcal{P}(Act \times \nu) \times \mathcal{P}(Act \times \nu) \xrightarrow{\delta_r} \mathcal{P}(Act \times (\nu \times \nu)) \xrightarrow{\mathsf{sel}} \mathcal{P}(Act \times (\nu \times \nu))$$

  onde

$$\mathsf{sel} = \mathsf{filter}_{\{0\}}$$

  filtra todas as falhas de sincronização, representados por 0. Note-se que a função δ_r é definida como

$$\delta_r\ \langle c_1, c_2\rangle \;=\; \{\langle a'\ \theta\ a, \langle p, p'\rangle\rangle |\ \langle a, p\rangle \in c_1 \ \wedge\ \langle a', p'\rangle \in c_2\}$$

  onde intervém a álgebra de interacção θ.

- **Paralelo**
  A composição paralela, intuitivamente, combina num único operador os efeitos de $|||$ e $\otimes$: o processo resultante exibe quer o comportamento de cada um dos argumentos independentes quer o que resulta da efectivação de sincronizações. Note-se que essa combinação é feita ao nível dos *genes* do anamorfismo. Portanto, o seu "gene" recorrerá aos genes de $|||$ e de $\otimes$. Formalmente,

$$| \;=\; [\![(\alpha_{|})]\!]$$

  onde

$$\alpha_{|} \;=\; \nu \times \nu \xrightarrow{\Delta} (\nu \times \nu) \times (\nu \times \nu) \xrightarrow{(\alpha_{|||}\times\alpha_{\otimes})} \mathcal{P}(Act \times (\nu \times \nu)) \times \mathcal{P}(Act \times (\nu \times \nu)) \xrightarrow{\cup} \mathcal{P}(Act \times (\nu \times \nu))$$

4.3 Animação Funcional

A definição coindutiva dos combinadores típicos de uma álgebra de processos, revista na secção anterior, permite uma codificação de um animador em HASKELL de forma quase directa. Nesta secção apresentamos os elementos essenciais desse animador para uma álgebra de processos particular — o CCS. Em apêndice, porém, são discutidas duas variantes à álgebra de interacção subjacente. Assim, tratam-se os casos em que

- a disciplina de interacção segue a convenção adoptada em CSP [Hoa85], em que apenas as acções com o mesmo nome sincronizam, *i.e.*, θ é aí definida como $a\theta a = a$, para toda a acção a.
- a disciplina adoptada capta a ideia de *co-ocorrência*: duas acções sincronizam pela realização independente mas simultânea de cada uma delas. Formalmente $a\theta b = (a, b)$, para toda a acção a.

Note-se que, com excepção do que concerne à álgebra de interacção, a estrutura básica do cálculo mantém-se a mesma em todos os três casos.

Comecemos por apresentar a declaração do tipo de dados coindutivo que modela as denotações dos processos. A declaração coindutiva é similar à usada no capítulo anterior, instanciando o tipo `Nu`. O functor capta a forma como os processos poderão ser observados: como *listas* (e não conjuntos, dada a impossibilidade de implementação efectiva de colecções não ordenadas) de pares ordenados, em que o primeiro elemento é uma acção e o segundo, o processo resultante dessa acção.

```
data Pr a x =  C [(a, x)]  deriving Show

obsProc :: Pr a x -> [(a, x)]
obsProc p = f
         where  (C f) = p

instance Functor (Pr a)
    where fmap f (C s) = C (map (id >< f) s)

type Proc a = Nu (Pr a)
```

Segue-se a especificação da *álgebra de interacção* correspondente à disciplina de sincronização específica do CCS. A estrutura das acções é definida como um

tipo indutivo a partir de um conjunto arbitrário de identificadores, sobre a qual se define a operação θ, aqui designada por `prodAct`. Para efectuar comparações torna-se necessário definir explicitamente uma noção de igualdade entre acções `eqAct` e a correspondente relação de ordem `leqAct`.

```
data Act l = A l | AC l | Nop | Tau | Id  deriving Show

instance (Eq a) => Eq (Act a) where    x == y  = eqAct x y

instance (Ord a) => Ord (Act a) where  x <= y  = leqAct x y

prodAct :: (Eq a) => Act a -> Act a -> Act a
prodAct Nop _        = Nop
prodAct _ Nop        = Nop
prodAct Id x         = x prodAct x Id = x
prodAct (A x) (AC y)  = if (x == y) then Tau else Nop
prodAct (AC x) (A y) = if (x == y) then Tau
                       else Nop prodAct a1 a2 = Nop

eqAct :: (Eq a) => Act a -> Act a -> Bool
eqAct (A a1) (A a2)   = (a1 == a2)
eqAct (AC a1) (AC a2) = (a1 == a2)
eqAct Nop Nop         = True
eqAct Tau Tau         = True
eqAct Id Id           = True

leqAct :: (Ord a) => Act a -> Act a -> Bool
leqAct (A a1) (A a2)   = (a1 <= a2)
leqAct (AC a1) (AC a2) = (a1 <= a2)
leqAct Nop _           = True
leqAct Tau _           = True
leqAct Id _            = True
```

Vejamos, agora, as definições dos combinadores dinâmicos.

```
nilP :: Proc (Act a)
nilP  =  Fin (C [])

preP :: Act a -> Proc (Act a) -> Proc (Act a)
preP act p = Fin (C [(act,p)])
```

```
sumP ::  Proc (Act a) -> Proc (Act a) -> Proc (Act a)
sumP p q  = Fin (C (pp ++ qq)) where
            (C pp) = (unFin p)
            (C qq) = (unFin q)
```

E, por fim, as dos combinadores estáticos.

```
rename :: ((Act a) -> (Act a)) -> Proc (Act a) -> Proc (Act a)
rename f p = ana phi p
             where phi p = C (map (f >< id) (obsP p))

restriction :: (Eq a) => (Proc (Act a), [Act a])
                  -> Proc (Act a)
restriction (p,la) = ana  phi p
                     where phi p = C (filter (obsP p) la)

interleaving :: (Proc (Act a), Proc (Act a)) -> Proc (Act a)
interleaving (p,q) = ana alphai (p,q)

syn :: (Eq a) => (Proc (Act a),Proc (Act a)) -> Proc (Act a)
syn (p, q) = ana alphap (p, q)

par :: (Eq a) => (Proc (Act a), Proc (Act a)) -> Proc (Act a)
par (p, q) = ana alpha (p, q)
     where alpha (p, q) =
       C ((obsProc (alphai (p,q))) ++ (obsProc (alphap (p,q))))
```

Ilustremos, agora, o processo de animação propriamente dito. Para o facilitar foi desenvolvido um visualizador da evolução dos processos que inclui a função `vP` sobre um tipo paramétrico `ShowProc`.

```
data ShowProc a  =  V [(a, ShowProc a)]  deriving Show

vP :: Proc a -> ShowProc a
vP (Fin (C l)) = V (vPaux l)

vPaux :: [(a, Proc a)] -> [(a, ShowProc a)]
vPaux []        = []
vPaux ((a,p):t) = [(a, (vP p))] ++ (vPaux t)
```

Ilustram-se, de seguida, exemplificando o uso do animador, algumas leis típicas do CCS.

1. $(P + Q) + R = P + (Q + R)$

```
z1 = preP (A 2) nilP
z5 = preP (AC 5) nilP
z6 = preP (AC 6) nilP
z10 = sumP z1 z6
z31 = sumP z6 z5

ProcType> vP(z1)
V [(A 2,V [])]

ProcType> vP(z5)
V [(AC 5,V [])]

ProcType> vP(z6)
V [(AC 6,V [])]

ProcType> vP(z10)
V [(A 2,V[]),(AC 6,V [])]

ProcType> vP(z31)
V[(AC 6,V []),(AC 5,V [])]

ProcType> vP(sumP z10 z5)
V [(A 2,V[]),(AC 6,V []),(AC 5,V [])]

ProcType> vP(sumP z1 z31)
V [(A 2,V[]),(AC 6,V []),(AC 5,V [])]
```

2. $P + \mathsf{nil} = P$

```
z1  = preP (A 2) nilP
z28 = sumP z1 nilP
```

```
ProcType> vP(z1)
V [(A 2,V [])]

ProcType> vP(z28)
V [(A 2,V [])]
```

---

3. $P + P = P$

```
z1 = preP (A 2) nilP

ProcType> vP(z1)
V [(A 2,V [])]

ProcType> vP(sumP z1 z1)
V [(A 2,V []),(A 2,V [])]
```

---

4. $P + Q = Q + P$

```
z10 = sumP z1 z6
z12 = sumP z6 z1

ProcType> vP(z10)
V [(A2,V[]),(AC 6,V [])]

ProcType> vP(z12)
V [(AC6,V []),(A 2,V])]
```

---

5. $P||Q = Q||P$

```
z25 = par (z1, z5)
z26 = par (z5, z1)

ProcType> vP(z25)
```

```
V [(A 2,V [(AC 5,V [])]),(AC 5,V [(A 2,V [])])]

ProcType> vP(z26)
V [(AC 5,V [(A 2,V [])]),(A 2,V [(AC 5,V [])])]
```

---

6. nil$||P = P$

```
z1 = preP (A 2) nilP
z27 = par (nilP, z1)

ProcType> vP(z1)
V [(A 2,V [])]

ProcType> vP(z27)
V [(A 2,V [])]
```

---

7. $(P + Q)||R \neq P||R + Q||R$

```
z1  = preP (A 2) nilP
z5  = preP (AC 5) nilP
z6  = preP (AC 6) nilP
z10 = sumP z1 z6
z29 = par (z10,z5)

z30 = sumP (par z1 z5)(par z6 z5)

ProcType> vP(z29)
V [(A 2,V [(AC 5,V [])]),(AC 6,V [(AC 5,V
   [])]),(AC 5,V [(A 2,V []),(AC 6,V [])])]

ProcType> vP(z30)
V [(A 2,V [(AC 5,V [])]),(AC 5,V [(A 2,V
   [])]),(AC 6,V [(AC 5,V [])]),(AC 5,V [(AC 6 ,V [])])]
```

Ilustremos, agora, uma função que funciona como interface com o utilizador, uma vez que lhe permite escolher a acção a realizar. Vejamos a função construída `execProc`.

```
execProc :: (Show a) => Proc (Act a) -> IO () execProc (Fin (C l))
=
   case (length l) of
   0 -> putStrLn "Processo terminado"
   1 -> do putStrLn ("Accao " ++ show (fst . head $ l) ++ " executada.")
           execProc (snd . head $ l)
   _ -> do putStrLn . ((++) "Escolha a posicao da accao a executar")
        . show $ (map fst l)
           i <- getLine
           let p = posi (toInteiro i) l
           putStrLn ("Accao " ++ show (fst p) ++ " executada.")
           execProc (snd p)
```

Precisamos de saber determinar a posição em que se encontra a acção. Assim, desenvolvemos uma funçao que lê o número que indica essa posição e outra que procura a acção correspondente a essa posição.

```
toInteiro :: String -> Int
toInteiro = read

posi i = last . take i
```

Vejamos alguns exemplos.

```
ProcType> execProc z1
Accao A 2 executada
Processo terminado

ProcType> execProc z10
Escolha a posicao da accao a executar [A 2,AC 6]
1
Accao A 2 executada
Processo terminado
```

```
ProcType> execProc z10
Escolha a posicao da accao a executar [A 2,AC 6]
2
Accao AC 6 executada
Processo terminado
```

4.4 Novos Combinadores de Processsos

Nesta secção são introduzidos três novos combinadores ao núcleo base da família de álgebras de processos estudada em [Bar01b]. O primeiro é um operador de *internalização* de acções. O seu efeito é tornar não observáveis determinadas acções, o que o aproxima do combinador de *hiding* proposto em CSP [Hoa85]. Os dois últimos são operadores de *interrupção* que lidam com situações de excepção verificadas no decurso da execução dos processos. A dependência de informação contextual força a definição destes dois combinadores como *apomorfismos*. Assim, um dos objectivos deste estudo reside precisamente na análise de provas por cálculo envolvendo apomorfismos.

4.4.1 Internalização

O operador $\backslash_k$ tem como objectivo *internalizar* todas as ocorrências da acção k, mapeando-as na acção τ, que, em muitas álgebras de interacção, e em particular na do CCS, é tomada como a representação canónica de actividade interna, logo não observável.

Note-se, porém, que aquilo que o operador especifica é um mecanismo de mapeamento de uma acção noutra, pelo que a escolha de τ é irrelevante para a definição do seu comportamento. Formalmente,

$$\backslash_k \;=\; [\![(\alpha_k)]\!]$$

onde

$$\alpha_k \;=\; \nu \xrightarrow{\omega} \mathcal{P}(Act \times \nu) \xrightarrow{\mathcal{P}(\mathsf{sub}_k \times \mathsf{id})} \mathcal{P}(Act \times \nu)$$

em que $\mathsf{sub}_k \triangleq (=_k) \rightarrow \tau, \mathsf{id}$.

O combinador é facilmente generalizável a um conjunto K de acções: basta re-escrever a função sub como

$$\mathsf{sub}_K \triangleq \in_K \rightarrow \tau, \mathsf{id}$$

Vamos, agora, estudar a interacção deste combinador com a composição por entrelaçamento $|||$. Pretendemos provar o seguinte resultado de distribuição:

Lema 10

$$\backslash_k \cdot ||| = ||| \cdot \backslash_k \tag{4.1}$$

Prova. A igualdade (4.1) não está apresentada de forma a que seja possível aplicar-lhe directamente a lei de fusão associada aos anamorfismos, documentada no capítulo 3 desta dissertação. No entanto, sendo ω um isomorfismo podemos re-escrevê-la na forma

$$\omega \cdot \backslash_k \cdot ||| = \omega \cdot ||| \cdot \backslash_k \tag{4.2}$$

que é simplificável em termos dos genes dos operadores atendendo ao facto de tanto $|||$ como $\backslash_k$ terem sido definidos por coindução. Assim, comecemos por desenvolver o lado esquerdo da igualdade (4.2).

$$\begin{aligned}
& \omega \cdot \backslash_k \cdot ||| \\
= & \quad \{ \text{ por definição de } \omega \cdot \backslash_k \} \\
& \mathcal{P}(\mathsf{id} \times \backslash_k) \cdot \alpha_k \cdot ||| \\
= & \quad \{ \text{ por definição de } \alpha_k \} \\
& \mathcal{P}(\mathsf{id} \times \backslash_k) \cdot \mathcal{P}(\mathsf{sub}_k \times \mathsf{id}) \cdot \omega \cdot ||| \\
= & \quad \{ ||| \text{ é morfismo} \} \\
& \mathcal{P}(\mathsf{id} \times \backslash_k) \cdot \mathcal{P}(\mathsf{sub}_k \times \mathsf{id}) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_k \\
= & \quad \{ \text{ fucntorialidade e definição de } \alpha_{|||} \} \\
& \mathcal{P}(\mathsf{id} \times \backslash_k) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \mathcal{P}(\mathsf{sub}_k \times (\mathsf{id} \times \mathsf{id})) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot (\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega) \cdot \Delta \\
= & \quad \{ \cup \text{ natural} \} \\
& \mathcal{P}(\mathsf{id} \times \backslash_k) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot \mathcal{P}(\mathsf{sub}_k \times (\mathsf{id} \times \mathsf{id})) \times \mathcal{P}(\mathsf{sub}_k \times (\mathsf{id} \times \mathsf{id})) \cdot (\tau_r \times \tau_l) \\
& \cdot (\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega) \cdot \Delta \\
= & \quad \{ \tau_r \text{ e } \tau_l \text{ naturais } i.e. \tau_r \cdot (\mathsf{B}f \times g) = \mathsf{B}(f \times g) \cdot \tau_r \text{ e } \tau_l \cdot (f \times \mathsf{B}g) = \mathsf{B}(f \times g) \cdot \tau_l \text{ para } \mathsf{B} = \mathcal{P}(Act \times \mathsf{Id}) \} \\
& \mathcal{P}(\mathsf{id} \times \backslash_k) \cdot \cup \cdot (\tau_l \times \tau_r) \cdot (\mathcal{P}(\mathsf{sub}_k \times \mathsf{id}) \cdot \omega \times \mathsf{id}) \times (\mathsf{id} \times \mathcal{P}(\mathsf{sub}_k \times \mathsf{id}) \cdot \omega) \cdot \Delta \\
= & \quad \{ \text{ pela definição de } \alpha_k \} \\
& \mathcal{P}(\mathsf{id} \times \backslash_k) \cdot \cup \cdot (\tau_l \times \tau_r) \cdot ((\alpha_k \times \mathsf{id}) \times (\mathsf{id} \times \alpha_k)) \cdot \Delta
\end{aligned}$$

Vamos, agora, desenvolver o lado direito.

$$
\begin{aligned}
& \omega \cdot ||| \cdot (\backslash_k \cdot \backslash_k) \\
= & \quad \{ \text{ ||| é morfismo } \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_{|||} \cdot (\backslash_k \times \backslash_k) \\
= & \quad \{ \text{ por definição de } \alpha_{|||} \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \Delta \cdot (\backslash_k \times \backslash_k) \\
= & \quad \{ \Delta \text{ natural} \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot ((\backslash_k \times \backslash_k) \times (\backslash_k \times \backslash_k)) \cdot \Delta \\
= & \quad \{ \text{functorialidade} \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot (\omega \cdot \backslash_k \times \backslash_k) \times (\backslash_k \times \omega \cdot \backslash_k) \cdot \Delta \\
= & \quad \{ \backslash_k \text{ é morfismo } \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\mathcal{P}(\mathsf{id} \times \backslash_k) \cdot \alpha_k \times \backslash_k) \times (\backslash_k \times \mathcal{P}(\mathsf{id} \times \backslash_k) \cdot \alpha_k)) \cdot \Delta \\
= & \quad \{ \text{functorialidade} \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\mathcal{P}(\mathsf{id} \times \backslash_k) \times \backslash_k) \times (\backslash_k \times \mathcal{P}(\mathsf{id} \times \backslash_k))) \\
& \cdot ((\alpha_k \times \mathsf{id}) \times (\mathsf{id} \times \alpha_k)) \cdot \Delta \\
= & \quad \{ \tau_r \text{ e } \tau_l \text{ naturais} \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot \mathcal{P}(\mathsf{id} \times (\backslash_k \times \backslash_k)) \times \mathcal{P}(\mathsf{id} \times (\backslash_k \times \backslash_k)) \cdot (\tau_r \times \tau_l) \\
& \cdot ((\alpha_k \times \mathsf{id}) \times (\mathsf{id} \times \alpha_k)) \cdot \Delta \\
= & \quad \{ \cup \text{ natural} \} \\
& \mathcal{P}(\mathsf{id} \times (||| \cdot (\backslash_k \times \backslash_k))) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_k \times \mathsf{id}) \times (\mathsf{id} \times \alpha_k)) \cdot \Delta
\end{aligned}
$$

Não chegamos, de facto, a expressões iguais após este processo de simplificação de ambos os lados da igualdade (4.2). No entanto, concluimos que

$$\omega \cdot ||| \cdot (\backslash_k \times \backslash_k) = \mathcal{P}(\mathsf{id} \times (||| \cdot (\backslash_k \times \backslash_k))) \cdot \gamma$$

e

$$\omega \cdot \backslash_k \cdot ||| = \mathcal{P}(\mathsf{id} \times (\backslash_k \cdot |||)) \cdot \gamma$$

para a coalgebra

$$\gamma = \cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_k \times \mathsf{id}) \times (\mathsf{id} \times \alpha_k)) \cdot \Delta$$

Isto significa que tanto $||| \cdot (\backslash_k \times \backslash_k)$ como $\backslash_k \cdot |||$ são morfismos entre a coalgebra γ e a coalgebra final ω. De facto, o que as igualdades acima exprimem é a comuta-

tividade do diagrama

$$\begin{array}{ccc} \nu & \xrightarrow{\omega} & \mathcal{P}(Act \times \nu) \\ \uparrow & & \uparrow \\ \nu \times \nu & \xrightarrow[\gamma]{} & \mathcal{P}(Act \times (\nu \times \nu)) \end{array}$$

Como só pode haver um único morfismo nestas condições, eles coincidem.

□

Repare-se no tipo de prova por cálculo feita neste exemplo: não sendo possível uma aplicação directa da lei de fusão para anamorfismos, procurou-se mostrar que as duas formas de composição dos dois operadores podia ser definida como um anamorfismo para uma coalgebra comum γ, e logo, por unicidade, concluir que coincidiam. Note-se, ainda, que a coalgebra γ não foi postulada à partida, mas inferida por cálculo.

4.4.2 Interrupção

O combinador proposto nesta secção é uma forma de composição paralela que, ao contrário de $|||$, $\otimes$ ou $|$, prevê a possibilidade de se interromper e terminar, caso se verifiquem certas condições anómalas na sua interacção interna. A maneira de se modelarem tais condições anómalas, de forma suficientemente abstracta e genérica, consiste em associar a sua ocorrência a uma interacção particular que se arbitra designar por $*$. Assim, o combinador f_* interrompe se o resultado da interacção entre os dois processos argumentos fôr $*$.

O combinador é definido a partir do seguinte diagrama:

$$\begin{array}{ccc} \nu \times \nu & \xrightarrow{\varphi_*} & \mathcal{P}(Act \times ((\nu \times \nu) + \nu)) \\ {\scriptstyle f_*}\downarrow & & \downarrow{\scriptstyle \mathcal{P}(\mathsf{id} \times [f_*, \mathsf{id}])} \\ \nu & \xrightarrow{\omega} & \mathcal{P}(Act \times \nu) \end{array}$$

ou seja, é definido por uma apomorfismo:

$$f_* \;=\; \mathsf{apo}\,\varphi_* \tag{4.3}$$

sendo[5]

$$\begin{aligned}
\varphi_x \;=\; \nu \times \nu &\xrightarrow{\omega\times\omega} \mathcal{P}(Act \times \nu) \times \mathcal{P}(Act \times \nu) \\
&\xrightarrow{\tau_r} \mathcal{P}((Act \times \nu) \times \mathcal{P}(Act \times \nu)) \\
&\xrightarrow{\mathcal{P}\tau_l} \mathcal{P}\mathcal{P}((Act \times \nu) \times (Act \times \nu)) \\
&\xrightarrow{\cup} \mathcal{P}((Act \times \nu) \times (Act \times \nu)) \\
&\xrightarrow{\mathcal{P}\mathsf{m}} \mathcal{P}((Act \times Act) \times (\nu \times \nu)) \\
&\xrightarrow{\mathcal{P}(\theta\times\mathsf{id})} \mathcal{P}(Act \times (\nu \times \nu)) \\
&\xrightarrow{\mathcal{P}\mathsf{test}} \mathcal{P}(Act \times ((\nu \times \nu) + \nu))
\end{aligned}$$

onde

$$\mathsf{test} \;= \langle \pi_1, =_* \cdot \pi_1 \;\rightarrow\; \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \pi_2 \rangle$$

e $m : (A \times C) \times (B \times D) \longleftarrow (A \times B) \times (C \times D)$ é o isomorfismo natural que troca as posições relativas dos factores num produto.
Vamos, agora, estudar o cálculo com apomorfismos tentanto investigar a comutatividade deste operador, *i.e.*, a validade da seguinte equação

$$f_* \cdot \mathsf{s} = f_* \tag{4.4}$$

O primeiro passo nessa investigação é a derivação

$$\begin{aligned}
& f_* \cdot \mathsf{s} = f_* \\
\equiv \quad & \{ \text{ por definição de } f_* \} \\
& \mathsf{apo}\,\varphi_* \cdot \mathsf{s} \;=\; \mathsf{apo}\,\varphi_* \\
\Leftarrow \quad & \{ \text{ lei fusão-apo} \} \\
& \varphi_* \cdot \mathsf{s} \;=\; \mathcal{P}(\mathsf{id} \times (\mathsf{s} + \mathsf{id})) \cdot \varphi_*
\end{aligned}$$

Comecemos por desenvolver o primeiro membro desta última igualdade.

$$\begin{aligned}
& \varphi_* \cdot \mathsf{s} \\
= \quad & \{ \text{ por definição de } \varphi_* \} \\
& \mathcal{P}(\langle \pi_1, =_* \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \pi_2 \rangle) \cdot \mathcal{P}(\theta \times \mathsf{id}) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \cdot \tau_r \cdot \omega \times \omega \cdot \mathsf{s}
\end{aligned}$$

[5]Note-se que nesta definição os morfismos naturais τ_l e τ_r são tomados relativamente ao functor $\mathcal{P}$, e não ao functor $\mathcal{P}(Act \times \mathsf{Id})$ como nas restantes definições de combinadores de processos feitas nesta dissertação.

$$
\begin{aligned}
= & \quad \{ \text{ s natural } \} \\
& \mathcal{P}(\langle \pi_1, =_* \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \pi_2 \rangle) \cdot \mathcal{P}(\theta \times \mathsf{id}) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \cdot \tau_r \cdot \mathsf{s} \cdot \omega \times \omega \\
= & \quad \{ \ \tau_r \cdot \mathsf{s} = \mathcal{P}\mathsf{s} \cdot \tau_l \ \} \\
& \mathcal{P}(\langle \pi_1, =_* \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \pi_2 \rangle) \cdot \mathcal{P}(\theta \times \mathsf{id}) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \cdot \mathcal{P}\mathsf{s} \cdot \tau_l \cdot \omega \times \omega \\
= & \quad \{ \ \tau_l \cdot \mathsf{s} = \mathcal{P}\mathsf{s} \cdot \tau_r, \text{ functorialidade } \} \\
& \mathcal{P}(\langle \pi_1, =_* \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \pi_2 \rangle) \cdot \mathcal{P}(\theta \times \mathsf{id}) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\mathcal{P}\mathsf{s} \cdot \mathcal{P}\tau_r \cdot \tau_l \cdot \omega \times \omega \\
= & \quad \{ \ \cup \text{ natural } \} \\
& \mathcal{P}(\langle \pi_1, =_* \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \pi_2 \rangle) \cdot \mathcal{P}(\theta \times \mathsf{id}) \cdot \mathcal{P}\mathsf{m} \cdot \mathcal{P}\mathsf{s} \cdot \cup \cdot \mathcal{P}\tau_r \cdot \tau_l \cdot \omega \times \omega \\
= & \quad \{ \ \mathsf{m} \text{ natural: } \mathsf{m} \cdot \mathsf{s} = (\mathsf{s} \times \mathsf{s}) \cdot \mathsf{m} \ \} \\
& \mathcal{P}(\langle \pi_1, =_* \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \pi_2 \rangle) \cdot \mathcal{P}(\theta \times \mathsf{id}) \cdot \mathcal{P}(\mathsf{s} \times \mathsf{s}) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_r \cdot \tau_l \cdot \omega \times \omega \\
= & \quad \{ \ \mathcal{P}\tau_r \cdot \tau_l = \mathcal{P}\tau_l \cdot \tau_r, \text{ porque } \mathcal{P} \text{ é um monad comutativo [Koc72]; functorialidade } \} \\
& \mathcal{P}(\langle \pi_1, =_* \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \pi_2 \rangle) \cdot \mathcal{P}((\theta \cdot \mathsf{s}) \times \mathsf{s}) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \cdot \tau_r \cdot \omega \times \omega \\
= & \quad \{ \ \times\text{-fusão } \} \\
& \mathcal{P}(\langle \pi_1 \cdot ((\theta \cdot \mathsf{s}) \times \mathsf{s}) \,,\, (=_* \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \pi_2) \cdot ((\theta \cdot \mathsf{s}) \times \mathsf{s}) \rangle) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \\
& \cdot \tau_r \cdot \omega \times \omega \\
= & \quad \{ \ \times\text{-cancelamento, lei de fusão do condicional } \} \\
& \mathcal{P}(\langle \theta \cdot \mathsf{s} \cdot \pi_1 \,,\, (=_* \cdot \pi_1 \cdot ((\theta \cdot \mathsf{s}) \times \mathsf{s}) \rightarrow \iota_2 \cdot \mathsf{nil} \cdot ((\theta \cdot \mathsf{s}) \times \mathsf{s}), \iota_1 \cdot \pi_2 \cdot ((\theta \cdot \mathsf{s}) \times \mathsf{s})) \rangle) \\
& \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \cdot \tau_r \cdot \omega \times \omega \\
= & \quad \{ \ \times\text{-cancelamento, função constante } \} \\
& \mathcal{P}(\langle \theta \cdot \mathsf{s} \cdot \pi_1 \,,\, (=_* \cdot \theta \cdot \mathsf{s} \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \mathsf{s} \cdot \pi_2) \rangle) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \cdot \tau_r \cdot \omega \times \omega
\end{aligned}
$$

Desenvolvendo o segundo membro, vem

$$
\begin{aligned}
& \mathcal{P}(\mathsf{id} \times (\mathsf{s} + \mathsf{id})) \cdot \varphi_* \\
= & \quad \{ \text{ por definição de } \varphi_* \ \} \\
& \mathcal{P}(\mathsf{id} \times (\mathsf{s} + \mathsf{id})) \cdot \mathcal{P}(\langle \pi_1, =_* \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \pi_2 \rangle) \cdot \mathcal{P}(\theta \times \mathsf{id}) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \\
& \cdot \tau_r \cdot \omega \times \omega \\
= & \quad \{ \text{ functorialidade, } \times\text{-absorção } \} \\
& \mathcal{P}(\langle \pi_1, (\mathsf{s} + \mathsf{id}) \cdot (=_* \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \pi_2) \rangle) \cdot \mathcal{P}(\theta \times \mathsf{id}) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \cdot \tau_r \cdot \omega \times \omega \\
= & \quad \{ \text{ lei de fusão do condicional } \} \\
& \mathcal{P}(\langle \pi_1, =_* \cdot \pi_1 \rightarrow (\mathsf{s} + \mathsf{id}) \cdot \iota_2 \cdot \mathsf{nil}, (\mathsf{s} + \mathsf{id}) \cdot \iota_1 \cdot \pi_2 \rangle) \cdot \mathcal{P}(\theta \times \mathsf{id}) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \\
& \cdot \tau_r \cdot \omega \times \omega \\
= & \quad \{ \ +\text{-cancelamento } \} \\
& \mathcal{P}(\langle \pi_1, =_* \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \mathsf{s} \cdot \pi_2 \rangle) \cdot \mathcal{P}(\theta \times \mathsf{id}) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \cdot \tau_r \cdot \omega \times \omega \\
= & \quad \{ \text{ lei de fusão do condicional, functorialidade, } \times\text{-fusão } \}
\end{aligned}
$$

$$\mathcal{P}(\langle \theta \cdot \pi_1 \, , =_* \cdot \theta \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \mathsf{s} \cdot \pi_2 \rangle) \cdot \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \cdot \tau_r \cdot \omega \times \omega$$

Repare-se que os dois membros da equação não simplificaram na mesma expressão, pelo que não é possível a validade universal de (4.4). No entanto as expressões a que chegamos diferem apenas no facto de a operação θ sobre as acçoes ser aplicada por ordem diversa em cada uma delas. Torna-se, então, legítimo supôr que uma lei menos geral que (4.4), mas ainda assim, relevante e útil possa emergir deste cálculo. A questão merece ser discutida numa secção própria. É o que faremos de seguida.

4.4.3 Leis de Fusão Condicional

Vamos, nesta subsecção, reflectir sobre a situação com que deparamos na última prova. Recorde-se que o objectivo era a verificação da equação

$$f_* \cdot \mathsf{s} = f_* \tag{4.5}$$

por aplicação da lei de fusão dos apomorfismos, que simplificou na verificação da equação

$$\varphi_* \cdot \mathsf{s} \;=\; \mathcal{P}(\mathsf{id} \times (\mathsf{s} + \mathsf{id})) \cdot \varphi_* \tag{4.6}$$

Como sempre, a vantagem de utilizar fusão nestas provas consiste em deixar de ser necessário lidar directamente com as definições recursivas. Ao invés, passamos a manipular apenas os *genes* respectivos. O cálculo feito na subsecção anterior reduziu (4.6) a

$$\begin{array}{l} \mathcal{P}(\langle \theta \cdot \mathsf{s} \cdot \pi_1 \, , (=_* \cdot \theta \cdot \mathsf{s} \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \mathsf{s} \cdot \pi_2) \rangle) \cdot \gamma \\ = \\ \mathcal{P}(\langle \theta \cdot \pi_1 \, , (=_* \cdot \theta \cdot \pi_1 \rightarrow \iota_2 \cdot \mathsf{nil}, \iota_1 \cdot \mathsf{s} \cdot \pi_2) \rangle) \cdot \gamma \end{array}$$

onde

$$\gamma \;=\; \mathcal{P}\mathsf{m} \cdot \cup \cdot \mathcal{P}\tau_l \cdot \tau_r \cdot \omega \times \omega$$

Notemos agora que (4.6) é apenas válida se se verificar uma condição adicional que expresse a comutatividade de θ, *i.e.*,

$$\theta \cdot \mathsf{s} \;=\; \theta \tag{4.7}$$

A questão que é, então, legítimo colocar é a seguinte: qual a implicação desta validade condicional detectada ao nível dos genes na validade da equação inicial (4.5)? Antes de respondermos, tentemos formular a questão da forma mais geral

possível. Suponhamos que na prova de uma igualdade envolvendo apomorfismos concluímos a validade do antecedente da lei de fusão

$$\alpha \cdot f = \mathsf{T}(f + \mathsf{id}) \cdot \beta \quad \Rightarrow \quad \mathsf{apo}\,\alpha \cdot f \;=\; \mathsf{apo}\,\beta \tag{4.8}$$

depende de uma condição adicional Φ, *i.e.*,

$$\Phi \quad \Rightarrow \quad \alpha \cdot f \;=\; \mathsf{T}(f + \mathsf{id}) \cdot \beta \tag{4.9}$$

O que acontece é que Φ é formulada como uma condição *local* sobre os genes dos apomorfismos envolvidos, *i.e.*, sobre as continuações *imediatas* dos processos que se candidatem a satisfazer a equação que surge no lado direito da lei de fusão (4.8). Esta condição tem, por isso, de ser reforçada no sentido de se tornar recursivamente exigível a *todas* as continuações desses processos. Tecnicamente diremos que Φ deve ser transformado num predicado *invariante*: *i.e.*, um predicado que é mantido ao longo da dinâmica da coalgebra de interesse que, neste caso, é obviamente ω. Para podermos formular este resultado, necessitamos de recorrer a algumas noções de lógica modal interpretada sobre coalgebras. Necessitamos, em particular, da definição de invariante.

Um predicado $\phi : \mathbb{B} \longleftarrow U$ sobre o portador de uma T-coalgebra $\langle U, \gamma : \mathsf{T}\ U \longleftarrow U\rangle$ é dito um γ-*invariante* se é fechado para a dinâmica da coalgebra. Para formalisar esta ideia define-se um combinador de predicados $\bigcirc_\gamma$ [6]:

$$(\bigcirc_\gamma\ \phi)\ u \;\equiv\; \forall_{u' \in_{\mathsf{T}} \gamma\ u}\ .\ \phi\ u'$$

cujo significado é $\bigcirc_\gamma\ \phi$ se verifica em todos os estados cujos sucessores imediatos, caso existam, sob a coalgebra γ satisfazem ϕ. Então ϕ é um invariante sse

$$\phi \;\Rightarrow\; \bigcirc_\gamma\ \phi$$

ou, identificando predicados com os respectivos conjuntos característicos

$$\phi \subseteq \bigcirc_\gamma\ \phi$$

O combinador $\bigcirc_\gamma$ pode ser pode ser visto como o operador *next* da lógica modal [HC84]. A observação essencial é de que neste caso a estrutura sobre a qual o operador é interpretado é o sistema de transição correspondente à coalgebra γ[7].

[6] A notação $\in_{\mathsf{T}}$ designa a extensão da relação de pertença a functores regulares [MB04].

[7] Trata-se de um tópico importante na investigação sobre coalgebras. O leitor interessado é referido para [Mos99] e [Jac99], onde, em particular, se mostra como a lógica modal associada a uma colagebra é determinada pela sua forma registada no functor em relação ao qual se define.

A extensão infinitária de $\bigcirc_\gamma$ define o operador modal *sempre no futuro*. Assim, [Jac99] introduz $\Box_\gamma\ \phi$ como o *maior* ponto fixo da função $\lambda_x\ .\ \phi \cap \bigcirc_\gamma\ x$. Intuitivamente, $\Box_\gamma\ \phi$ significa 'ϕ verifica-se no estado presente e em todos os estados sucessores deste'. Um argumento simples [Jac99] mostra que $\Box_\gamma\ \phi$ é o maior invariante contido em ϕ. Neste conceito reside a chave para responder à questão que nos ocupa, conforme se mostra no seguinte lema.

Lema 11 *Sejam φ e β duas* T*-coalgebras e Φ um predicado sobre o portador de β. Nestas condições verificam-se as seguintes leis de* fusão condicional*:*

$$(\Phi \Rightarrow (\alpha \cdot h = \mathsf{T}\ h \cdot \beta)) \ \Rightarrow\ (\Box_\beta\ \Phi \Rightarrow ([\![\alpha]\!]_\mathsf{T} \cdot h = [\![\beta]\!]_\mathsf{T})) \tag{4.10}$$

e

$$(\Phi \Rightarrow (\alpha \cdot h = \mathsf{T}(h + \mathsf{id}) \cdot \beta)) \ \Rightarrow\ (\Box_\beta\ \Phi \Rightarrow (\mathsf{apo}_\alpha \cdot h = \mathsf{apo}_\beta \mathsf{T})) \tag{4.11}$$

Prova. A lei (4.10) foi provada em [Bar01b]. Vejamos, aqui, o caso dos apomorfismos demonstrando (4.11). Seja X o conjunto portador de β e i_Φ a inclusão em X do seu subconjunto classificado pelo predicado Φ, *i.e.*, $\Phi \cdot i_\Phi = \underline{true}\bullet!$. Recorde-se que qualquer β-invariante induz uma subcoalgebra β', o que torna $i_{\Box_\beta\ \Phi}$ um morfismo entre coalgebras de β' para β. Então,

$$\begin{array}{ll}
 & \Phi \Rightarrow \alpha \cdot h = \mathsf{T}(h + \mathsf{id}) \cdot \beta \\
\equiv & \quad \{ \text{definição da inclusão } i_\Phi \} \\
 & \alpha \cdot h \cdot i_\Phi = \mathsf{T}(h + \mathsf{id}) \cdot \beta \cdot i_\Phi \\
\Rightarrow & \quad \{ \Box_\beta \Phi \subseteq \Phi \} \\
 & \alpha \cdot h \cdot i_{\Box_\beta \Phi} = \mathsf{T}(h + \mathsf{id}) \cdot \beta \cdot i_{\Box_\beta \Phi} \\
\equiv & \quad \{ i_{\Box_\beta \Phi} \text{ é um morfismo de } \beta' \text{ para } \beta \} \\
 & \alpha \cdot h \cdot i_{\Box_\beta \Phi} = \mathsf{T}(h + \mathsf{id}) \cdot \mathsf{T}(i_{\Box_\beta \Phi}) \cdot \beta' \\
\equiv & \quad \{ \text{functorialidade} \} \\
 & \alpha \cdot h \cdot i_{\Box_\beta \Phi} = \mathsf{T}((h + \mathsf{id}) \cdot i_{\Box_\beta \Phi}) \cdot \beta' \\
\equiv & \quad \{ \text{fusão para apomorfismos} \} \\
 & \mathsf{apo}\, \alpha \cdot h \cdot i_{\Box_\beta} = \mathsf{apo}\, \beta' \\
\equiv & \quad \{ i_{\Box_\beta \Phi} \text{ é morfismo entre coalgebras} \}
\end{array}$$

$$\begin{array}{ll} & \mathsf{apo}\,\alpha \cdot h \cdot i_{\Box_\beta \Phi} = \mathsf{apo}\,\beta \cdot i_{\Box_\beta \Phi} \\ \equiv & \quad \{ \text{definição da inclusão } i_{\Box_\beta \Phi} \} \\ & \Box_\beta \Phi \Rightarrow (\mathsf{apo}\,\alpha \cdot h = \mathsf{apo}\,\beta) \end{array}$$

□

Voltando, agora, ao exemplo em mãos, note-se que, neste caso, o predicado utilizado, equação (4.7), não envolve os estados, mas apenas as acções. Assim, temos que

$$\Box_\omega (\theta \cdot \mathsf{s} = \theta) \;=\; (\theta \cdot \mathsf{s} = \theta)$$

o que significa ser este o predicado a usar como antecedente da equação (4.6). O resultado final, que acabamos de estabelecer para este exemplo ao longo das duas últimas subsecções, é, pois, o seguinte:

$$(\theta \cdot \mathsf{s} = \theta) \;\Rightarrow\; f_* \cdot \mathsf{s} = f_* \tag{4.12}$$

4.4.4 Recuperação

O operador f_x visa modelar situações de recuperação de falhas na execução de um sistema[8]. Intuitivamente, o combinador permite a execução do primeiro processo argumento até que um erro seja detectado. Por convenção a ocorrência de um erro é modelada pela execução de uma acção especial x. Nessas circunstâncias, o processo corrente é terminado passando o controlo para o segundo argumento. Este processo, fornecido em segundo argumento, constitui a abstracção do código de recuperação do sistema. O combinador é definido a partir do seguinte diagrama:

$$\begin{array}{ccc} \nu \times \nu & \xrightarrow{\varphi_x} & \mathcal{P}(Act \times ((\nu \times \nu) + \nu)) \\ {\scriptstyle f_x}\downarrow & & \downarrow{\scriptstyle \mathcal{P}(\mathsf{id} \times [f_x, \mathsf{id}])} \\ \nu & \xrightarrow{\omega} & \mathcal{P}(Act \times \nu) \end{array}$$

Formalmente,

$$f_x \;=\; \mathsf{apo}\,\varphi_x$$

[8] Apesar de aqui o abordarmos a um nível muito elevado de abstração, cumpre sublinhar que a *tolerância a falhas* é um problema muito importante em engenharia dos sistemas informáticos.

onde

$$\begin{aligned}\varphi_x \;=\; & \nu \times \nu \xrightarrow{\omega \times \mathsf{id}} \mathcal{P}(Act \times \nu) \times \nu \\ & \xrightarrow{\mathsf{t}_x \cdot \pi_1 \to \iota_1, \iota_2 \cdot \pi_2} \mathcal{P}(Act \times \nu) \times \nu + \nu \\ & \xrightarrow{\tau_r + \omega} \mathcal{P}(Act \times (\nu \times \nu)) + \mathcal{P}(Act \times \nu) \\ & \xrightarrow{[\mathcal{P}(\mathsf{id} \times \iota_1), \mathcal{P}(\mathsf{id} \times \iota_2)]} \mathcal{P}(Act \times (\nu \times \nu + \nu))\end{aligned}$$

onde $\mathsf{t}_x : \mathbb{B} \longleftarrow \mathcal{P}(Act \times \nu)$ é definido por

$$\mathsf{t}_x \;=\; \notin_x \cdot \mathcal{P}\pi_1$$

Continuemos a investigar o raciocínio coindutivo provando uma nova propriedade condicional. Intuitivamente, caso não seja detectado erro no primeiro processo, *i.e.*, uma falha na sua computação, é apenas este que é executado. Dito de outro modo, um processo que executa num ambiente tolerante a falhas, na ausência de erros detectados, comporta-se exactamente como se fosse executado de forma autónoma. Formalmente,

Lema 12

$$\Box_\omega (\notin_x \cdot \mathcal{P}\pi_1) \quad \Rightarrow \quad f_x \;=\; \pi_1$$

Prova. Repare-se que o predicado $\notin_x \cdot \mathcal{P}\pi_1$ apenas estabelece a não ocorrência da acção x nos sucessores imediatos de cada estado. É, assim, suficiente para verificar o antecedente da lei de fusão para apomorfismos, como se mostra no cálculo seguinte.

$$\begin{aligned}
& \mathcal{P}(\mathsf{id} \times (\pi_1 + \mathsf{id})) \cdot \varphi_x \\
= & \quad \{ \text{por definição de } \mathsf{t}_x \text{ e assumindo a hipótese } \notin_x \cdot \mathcal{P}\pi_1 \} \\
& \mathcal{P}(\mathsf{id} \times (\pi_1 + \mathsf{id})) \cdot [\mathcal{P}(\mathsf{id} \times \iota_1), \mathcal{P}(\mathsf{id} \times \iota_2)] \cdot (\tau_r + \omega) \cdot \iota_1 \cdot \omega \times \mathsf{id} \\
= & \quad \{ \tau_r + \omega = [\iota_1 \cdot \tau_r, \iota_2 \cdot \omega] \} \\
& \mathcal{P}(\mathsf{id} \times (\pi_1 + \mathsf{id})) \cdot [\mathcal{P}(\mathsf{id} \times \iota_1), \mathcal{P}(\mathsf{id} \times \iota_2)] \cdot [\iota_1 \cdot \tau_r, \iota_2 \cdot \omega] \cdot \iota_1 \cdot \omega \times \mathsf{id} \\
= & \quad \{ + \text{-cancelamento} \} \\
& \mathcal{P}(\mathsf{id} \times (\pi_1 + \mathsf{id})) \cdot [\mathcal{P}(\mathsf{id} \times \iota_1), \mathcal{P}(\mathsf{id} \times \iota_2)] \cdot \iota_1 \cdot \tau_r \cdot \omega \times \mathsf{id} \\
= & \quad \{ + \text{-cancelamento} \} \\
& \mathcal{P}(\mathsf{id} \times (\pi_1 + \mathsf{id})) \cdot \mathcal{P}(\mathsf{id} \times \iota_1) \cdot \tau_r \cdot \omega \times \mathsf{id}
\end{aligned}$$

$$
\begin{array}{ll}
= & \{\ \mathcal{P}\text{ functor}\ \} \\
& \mathcal{P}(\mathsf{id} \times (\pi_1 + \mathsf{id}) \cdot \iota_1) \cdot \tau_r \cdot \omega \times \mathsf{id} \\
= & \{\ +\text{-cancelamento, functorialidade}\ \} \\
& \mathcal{P}(\mathsf{id} \times \iota_1) \cdot \mathcal{P}(\mathsf{id} \times \pi_1) \cdot \tau_r \cdot \omega \times \mathsf{id} \\
= & \{\ \mathcal{P}(\mathsf{id} \times \pi_1) \cdot \tau_r = \pi_1\ [\text{Bar01a}]\ \} \\
& \mathcal{P}(\mathsf{id} \times \iota_1) \cdot \pi_1 \cdot \omega \times \mathsf{id} \\
= & \{\ f \times g = \langle f, g \rangle, \times\text{-cancelamento}\ \} \\
& \mathcal{P}(\mathsf{id} \times \iota_1) \cdot \omega \cdot \pi_1
\end{array}
$$

Em rigor estaríamos à espera de ter provado

$$\mathcal{P}(\mathsf{id} \times (\pi_1 + \mathsf{id})) \cdot \varphi_x \ = \ \omega \cdot \pi_1$$

Mas mostramos, de facto, que

$$\mathcal{P}(\mathsf{id} \times (\pi_1 + \mathsf{id})) \cdot \varphi_x \ = \ \mathcal{P}(\mathsf{id} \times \iota_1) \cdot \omega \cdot \pi_1$$

O resultado não surpreende: em verdade o factor adicional $\mathcal{P}(\mathsf{id} \times \iota_1)$ no segundo membro está apenas a garantir a compatibilidade de tipos entre os dois membros. O que é importante é reter que a verificação foi feita assumindo, logo no primeiro passo, a hipótese $\notin_x \cdot \mathcal{P}\pi_1$. Logo, pela lei da fusão condicional do lema 11, concluímos conforme esperado.

□

4.5 Transposição Relacional

4.5.1 Coalgebras e Sistemas de Transição

É objectivo desta secção explorar uma aproximação alternativa ao raciocínio em álgebras de processos baseada no cálculo relacional [BH93, BM97]. O ponto de partida, que simultaneamente estabelece a relação com as subsecções anteriores, é a transposição de coalgebras para o functor $\mathcal{P}(Act \times \mathsf{Id})$ para relações binárias. Sendo essa transposição um isomorfismo, torna-se possível raciocinar em ambos os mundos, sem qualquer perda de detalhe e tirando partido das respectivas estruturas algébricas. Comecemos, portanto, por clarificar essa transposição relacionando formalmente sistemas de transição e coalgebras para o functor acima indicado[9].

[9]Por uma questão de economia notacional, abreviaremos, no que se segue, *Act* para *A*.

Definição 17 *Um sistema de transição etiquetado é uma seta*

$$_{\alpha}\longleftarrow : A \times U \longleftarrow U$$

na categoria Rel *das relações binárias,* i.e.*, uma relação* $_{\alpha}\longleftarrow \subseteq (A \times U) \times U$*, onde U é o conjunto portador do espaço de estados e A é o conjunto das etiquetas que testemunham as transições de estado, usualmente designado por alfabeto.*

Por *transposição* um sistema de transição $_{\alpha}\longleftarrow$ pode ser convertido numa coalgebra

$$\alpha : \mathcal{P}(A \times U) \longleftarrow U$$

em Set para o funtor $\mathsf{T}X = \mathcal{P}(A \times X)$. Na verdade o operador de transposição em causa (Λ) introduz, pela sua caracterização *universal*, não apenas uma equivalência entre estas duas representações, mas também o mecanismo de conversão entre elas:

$$\alpha = \Lambda\ {}_{\alpha}\longleftarrow \quad \equiv \quad {}_{\alpha}\longleftarrow = \in \cdot \alpha \tag{4.13}$$

Esta equivalência estende-se ao nível dos morfismos. Recordemos que um morfismo $h : \beta \longleftarrow \alpha$ entre coalgebras α e β é uma função entre os respectivos espaços de estados que preserva a dinâmica da coalgebra de partida, *i.e.*, que verifica

$$\mathcal{P}(\mathsf{id} \times h) \cdot \alpha \ = \ \beta \cdot h \tag{4.14}$$

Qual a contrapartida relacional desta equação? A resposta é confirmada por um simples cálculo.

Lema 13 *Uma função* $h : V \longleftarrow U$ *é um morfismo entre duas* $\mathcal{P}(A \times \mathsf{Id})$*-coalgebras* α *e* β *definidas sobre U e V, respectivamente, sse satisfizer a igualdade*

$$(\mathsf{id} \times h) \cdot {}_{\alpha}\longleftarrow \quad = \quad {}_{\beta}\longleftarrow \cdot h \tag{4.15}$$

cuja formulação envolve os sistemas de transição $_{\alpha}\longleftarrow$ *e* $_{\beta}\longleftarrow$ *correspondentes.*

Prova.

$$
\begin{array}{ll}
 & (\mathsf{id} \times h) \cdot {}_{\alpha}\longleftarrow \;=\; {}_{\beta}\longleftarrow \cdot h \\
\equiv & \quad \{ \text{ transposição é isomorfismo } \} \\
 & \Lambda((\mathsf{id} \times h) \cdot {}_{\alpha}\longleftarrow) \;=\; \Lambda({}_{\beta}\longleftarrow \cdot h) \\
\equiv & \quad \{ \Lambda(f \cdot R) = \mathcal{P}f \cdot \Lambda R \;\text{ e }\; \Lambda(R \cdot f) = \Lambda R \cdot f \} \\
 & \mathcal{P}(\mathsf{id} \times h) \cdot \Lambda({}_{\alpha}\longleftarrow) \;=\; \Lambda({}_{\beta}\longleftarrow) \cdot h \\
\equiv & \quad \{ \text{ definição de } {}_{\alpha}\longleftarrow \} \\
 & \mathcal{P}(\mathsf{id} \times h) \cdot \Lambda(\in \cdot \alpha) \;=\; \Lambda(\in \cdot \beta) \cdot h \\
\equiv & \quad \{ \Lambda(R \cdot f) = \Lambda R \cdot f \} \\
 & \mathcal{P}(\mathsf{id} \times h) \cdot \Lambda(\in) \cdot \alpha \;=\; \Lambda(\in) \cdot \beta \cdot h \\
\equiv & \quad \{ \Lambda(\in) = \mathsf{id} \} \\
 & \mathcal{P}(\mathsf{id} \times h) \cdot \alpha \;=\; \beta \cdot h
\end{array}
$$

□

Uma forma muito comum de representar um sistema de transição ${}_{\alpha}\longleftarrow$ sobre um alfabeto A é através de uma família A-indexada de relações binárias

$$ {}_{\alpha}\overset{a}{\longleftarrow} : U \longleftarrow U \tag{4.16} $$

para todo o $a \in A$. Para simplificar a escrita, o converso desta relação será representado por

$$ ({}_{\alpha}\overset{a}{\longleftarrow})^{\circ} = \overset{a}{\longrightarrow}_{\alpha} \tag{4.17} $$

Nesta representação a equação (4.15) corresponde também ela a uma família A-indexada de equações:

$$ h \cdot {}_{\alpha}\overset{a}{\longleftarrow} \;=\; {}_{\beta}\overset{a}{\longleftarrow} \cdot h \tag{4.18} $$

Como qualquer igualdade entre relações, para cada $a \in A$, (4.18) pode ser decomposta na conjunção das inclusões:

$$ h \cdot {}_{\alpha}\overset{a}{\longleftarrow} \;\subseteq\; {}_{\beta}\overset{a}{\longleftarrow} \cdot h \tag{4.19} $$

$$ {}_{\beta}\overset{a}{\longleftarrow} \cdot h \;\subseteq\; h \cdot {}_{\alpha}\overset{a}{\longleftarrow} \tag{4.20} $$

que, uma vez introduzidas variáveis e convertidas em predicados, adquirem a forma mais familiar:

$$\forall_{u,u' \in U} \,.\, u' \;{}_{\alpha}\overset{a}{\longleftarrow}\; u \;\Rightarrow\; h\,u' \;{}_{\beta}\overset{a}{\longleftarrow}\; h\,u \tag{4.21}$$

$$\forall_{u \in U, v' \in V} \,.\, v' \;{}_{\beta}\overset{a}{\longleftarrow}\; h\,u \;\Rightarrow\; \exists_{u' \in U}.\; u' \;{}_{\alpha}\overset{a}{\longleftarrow}\; u \;\wedge\; v' = h\,u' \tag{4.22}$$

porque:

Prova.

$$\begin{array}{ll} & h \cdot {}_{\alpha}\overset{a}{\longleftarrow} \;\subseteq\; {}_{\beta}\overset{a}{\longleftarrow} \cdot h \\ \equiv & \quad \{ \text{ lei de } \textit{shunting} \text{ (2.39)} \} \\ & {}_{\alpha}\overset{a}{\longleftarrow} \;\subseteq\; h^{\circ} \cdot {}_{\beta}\overset{a}{\longleftarrow} \cdot h \\ \equiv & \quad \{ \text{ introdução de variáveis} \} \\ & \forall_{u,u' \in U} \,.\, u' \;{}_{\alpha}\overset{a}{\longleftarrow}\; u \;\Rightarrow\; u' \,(h^{\circ} \cdot {}_{\beta}\overset{a}{\longleftarrow} \cdot h)\, u \\ \equiv & \quad \{ \text{ lei (2.37)} \} \\ & \forall_{u,u' \in U} \,.\, u' \;{}_{\alpha}\overset{a}{\longleftarrow}\; u \;\Rightarrow\; h\,u' \;{}_{\beta}\overset{a}{\longleftarrow}\; h\,u \end{array}$$

e

$$\begin{array}{ll} & {}_{\beta}\overset{a}{\longleftarrow} \cdot h \;\subseteq\; h \cdot {}_{\alpha}\overset{a}{\longleftarrow} \\ \equiv & \quad \{ \text{ introdução de variáveis} \} \\ & \forall_{u \in U, v' \in V} \,.\, v' \,({}_{\beta}\overset{a}{\longleftarrow} \cdot h)\, u \;\Rightarrow\; v' \,(h \cdot {}_{\alpha}\overset{a}{\longleftarrow})\, u \\ \equiv & \quad \{ \text{ lei (2.37) e definição de composição de relações} \} \\ & \forall_{u \in U, v' \in V} \,.\, v' \;{}_{\beta}\overset{a}{\longleftarrow}\; h\,u \;\Rightarrow\; \exists_{u' \in U}.\; u' \;{}_{\alpha}\overset{a}{\longleftarrow}\; u \;\wedge\; v' = h\,u') \end{array}$$

□

Tomadas conjuntamente estas equações afirmam que, não apenas a dinâmica de α, representada pelo sistema de transição ${}_{\alpha}\longleftarrow$ correspondente, é *preservada* por h (4.19), mas adicionalmente a dinâmica de β é *reflectida* para trás através do mesmo h (4.20). O que nos conduz directamente às noções de *simulação* e *bissimulação* que revisitaremos na próxima secção. Antes, porém, registemos um resultado técnico relacionado com a extensão da relação de transição a sequências de acções, que será útil mais tarde. Comecemos por definir

Definição 18 *A extensão de* $_\alpha\longleftarrow^{a}$ *uma sequência* $s \in Act^*$ *de acções é definida por recursão como*

$$_\alpha\longleftarrow^{[]} = \emptyset \tag{4.23}$$

$$_\alpha\longleftarrow^{a^t} = {}_\alpha\longleftarrow^{t} \cdot {}_\alpha\longleftarrow^{a} \tag{4.24}$$

Então,

Lema 14 *Sejam* $\alpha : \mathcal{P}(Act \times U) \longleftarrow U$ *e* $\beta : \mathcal{P}(Act \times V) \longleftarrow V$ *coalgebras para* $\mathcal{P}(Act \times \mathsf{Id})$ *e* $h : \beta \longleftarrow \alpha$ *um morfismo entre elas. Então,*

$$h \cdot {}_\alpha\longleftarrow^{s} = {}_\beta\longleftarrow^{s} \cdot h$$

para qualquer sequência $s \in Act^*$.

Prova. A prova prossegue por indução sobre o comprimento de s. Por 4.23 o caso $s = []$ é trivial. Tomemos, agora, como hipótese de indução a igualdade $h \cdot {}_\alpha\longleftarrow^{\mathsf{tl}s} = {}_\beta\longleftarrow^{\mathsf{tl}s} \cdot h$. Então,

$$\begin{array}{ll} & h \cdot {}_\alpha\longleftarrow^{a^t} \\ \equiv & \quad \{ \text{ (4.24) } \} \\ & h \cdot {}_\alpha\longleftarrow^{t} \cdot {}_\alpha\longleftarrow^{a} \\ \equiv & \quad \{ \text{ hipótese de indução } \} \\ & {}_\beta\longleftarrow^{t} \cdot h \cdot {}_\alpha\longleftarrow^{a} \\ \equiv & \quad \{ \text{ } h \text{ é morfismo } \} \\ & {}_\beta\longleftarrow^{t} \cdot {}_\beta\longleftarrow^{a} \cdot h \\ \equiv & \quad \{ \text{ (4.24) } \} \\ & {}_\beta\longleftarrow^{a^t} \cdot h \end{array}$$

□

4.5.2 Simulação e Bissimulação

Definição 19 *Dados dois sistemas de transição* $_\alpha\longleftarrow : U \times A \longleftarrow U$ *e* $_\beta\longleftarrow : V \times A \longleftarrow V$ *definidos sobre um mesmo conjunto de etiquetas A, uma* simulação *de* $_\alpha\longleftarrow$ *em* $_\beta\longleftarrow$ *é uma relação* $S : V \longleftarrow U$ *tal que*

$$\forall_{a \in A} \forall_{u \in U, v \in V} \,.\; v S u \;\Rightarrow\; (\forall_{u' \in U} \,.\; u' \,{}_\alpha\longleftarrow^{a}\, u \Rightarrow (\exists_{v' \in V} \,.\; v' \,{}_\beta\longleftarrow^{a}\, v \wedge v' S u')) \tag{4.25}$$

A definição pode ser formulada no cálculo relacional, onde a fórmula de primeira ordem é convertida numa expressão puramente algébrica. Assim,

Lema 15 *Uma relação $S : V \longleftarrow U$ é uma* simulação *de ${}_{\alpha}\longleftarrow$ em ${}_{\beta}\longleftarrow$ sse, para todo o $a \in A$*

$$S \cdot \xrightarrow{a}_{\alpha} \subseteq \xrightarrow{a}_{\beta} \cdot S \tag{4.26}$$

Prova.

$$\begin{array}{ll}
 & \forall_{a\in A}\forall_{u\in U,v\in V} .\ vS\,u \ \Rightarrow\ (\forall_{u'\in U} .\ u'\ {}_{\alpha}\xleftarrow{a}\ u \Rightarrow (\exists_{v'\in V} .\ v'\ {}_{\beta}\xleftarrow{a}\ v \wedge v'S\,u')) \\
\equiv & \quad \{\ \text{(4.17) e definição de composição de relações}\ \} \\
 & \forall_{a\in A}\forall_{u\in U,v\in V} .\ vS\,u \ \Rightarrow\ (\forall_{u'\in U} .\ u \xrightarrow{a}_{\alpha} u' \Rightarrow v\,(\xrightarrow{a}_{\beta} \cdot S)\,u' \\
\equiv & \quad \{\ \text{definição de divisão relacional à esquerda}\ \} \\
 & \forall_{a\in A}\forall_{u\in U,v\in V} .\ vS\,u \ \Rightarrow\ v\,((\xrightarrow{a}_{\beta} \cdot S)/\xrightarrow{a}_{\alpha})\,u \\
\equiv & \quad \{\ \text{eliminação de variáveis}\ \} \\
 & S \subseteq (\xrightarrow{a}_{\beta} \cdot S)/\xrightarrow{a}_{\alpha} \\
\equiv & \quad \{\ \text{conecção de Galois: } (\cdot R) \dashv (/R)\ \} \\
 & S \cdot \xrightarrow{a}_{\alpha} \subseteq \xrightarrow{a}_{\beta} \cdot S
\end{array}$$

□

Lema 16

1. *A relação vazia ($\bot$) e a identidade no espaço de estados (*id*) são simulações.*
2. *A composição de duas simulações é uma simulação.*
3. *A união de duas simulações é uma simulação.*

Prova.

1. Consideremos um sistema de transição ${}_{\alpha}\longleftarrow$ sobre um alfabeto A e um espaço de estados U. Então,

$$\begin{array}{ll}
 & \bot \cdot \xrightarrow{a}_{\alpha} \subseteq \xrightarrow{a}_{\beta} \cdot \bot \ \wedge\ \mathsf{id} \cdot \xrightarrow{a}_{\alpha} \subseteq \xrightarrow{a}_{\alpha} \cdot \mathsf{id} \\
\equiv & \quad \{\ \bot \text{ e id são, respectivamente, elemento absorvente e neutro da composição de relações}\ \} \\
 & \mathsf{true}
\end{array}$$

2. Consideremos os seguintes sistemas de transição ${}_{\beta}\longleftarrow : V \times A \longleftarrow V$, ${}_{\gamma}\longleftarrow : Z \times A \longleftarrow Z$ e ${}_{\alpha}\longleftarrow : U \times A \longleftarrow U$, onde estão definidas as simulações $S : {}_{\beta}\longleftarrow \longleftarrow {}_{\gamma}\longleftarrow$ e $R : {}_{\gamma}\longleftarrow \longleftarrow {}_{\alpha}\longleftarrow$. Então,

$$
\begin{array}{ll}
 & (S \cdot R)\cdot \stackrel{a}{\longrightarrow}_{\alpha} \subseteq \stackrel{a}{\longrightarrow}_{\beta} \cdot (S \cdot R) \\
\Leftarrow & \quad \{\ S\cdot \stackrel{a}{\longrightarrow}_{\gamma} \subseteq \stackrel{a}{\longrightarrow}_{\beta} \cdot S,\ \cdot\text{-assoc, monotonia}\ \} \\
 & (S \cdot R)\cdot \stackrel{a}{\longrightarrow}_{\alpha} \subseteq S\cdot \stackrel{a}{\longrightarrow}_{\gamma} \cdot R \\
\Leftarrow & \quad \{\ R\cdot \stackrel{a}{\longrightarrow}_{\alpha} \subseteq \stackrel{a}{\longrightarrow}_{\gamma} \cdot R,\ \cdot\text{-assoc, monotonia}\ \} \\
 & (S \cdot R)\cdot \stackrel{a}{\longrightarrow}_{\alpha} \subseteq (S \cdot R)\cdot \stackrel{a}{\longrightarrow}_{\alpha} \\
\equiv & \quad \{\ \mathit{trivial}\ \} \\
 & \mathsf{true}
\end{array}
$$

3. Consideremos, agora, os sistemas ${}_{\beta}\longleftarrow : V\times A \longleftarrow V$ e ${}_{\alpha}\longleftarrow : U\times A \longleftarrow U$ entre os quais estão definidas as simulações $S : {}_{\beta}\longleftarrow \longleftarrow {}_{\alpha}\longleftarrow$ e $R : {}_{\beta}\longleftarrow \longleftarrow {}_{\alpha}\longleftarrow$. Então,

$$
\begin{array}{ll}
 & (S \cup R)\cdot \stackrel{a}{\longrightarrow}_{\alpha} \subseteq \stackrel{a}{\longrightarrow}_{\beta} \cdot (S \cup R) \\
\equiv & \quad \{\ (R\cdot) \text{ e } (\cdot R) \text{ preservam } \cup \text{ por serem adjuntos inferiores em } (R\cdot) \dashv (R\backslash) \text{ e } (\cdot R) \dashv (/R), \text{ respectivamente}\ \} \\
 & (S \cdot \stackrel{a}{\longrightarrow}_{\alpha} \ \cup R\cdot \stackrel{a}{\longrightarrow}_{\alpha}) \subseteq (\stackrel{a}{\longrightarrow}_{\beta} \cdot S \cup \stackrel{a}{\longrightarrow}_{\beta} \cdot R) \\
\Leftarrow & \quad \{\ \text{definição de } \cup\ \} \\
 & S \cdot \stackrel{a}{\longrightarrow}_{\alpha} \subseteq \stackrel{a}{\longrightarrow}_{\beta} \cdot S \ \wedge\ R\cdot \stackrel{a}{\longrightarrow}_{\alpha} \subseteq \stackrel{a}{\longrightarrow}_{\beta} \cdot R \\
\equiv & \quad \{\ \text{hipótese}\ \} \\
 & \mathsf{true}
\end{array}
$$

□

Definição 20 *Uma relação $S : V \longleftarrow U$ entre os espaços de estados de dois sistemas de transição ${}_{\alpha}\longleftarrow : U \times A \longleftarrow U$ e ${}_{\beta}\longleftarrow : V \times A \longleftarrow V$ definidos sobre um mesmo conjunto de etiquetas A é uma* bissimulação *sse tanto S como S° forem simulações.*

Esta definição leva-nos à seguinte caracterização:

Lema 17 *Uma relação $S : V \longleftarrow U$ é uma* bissimulação *entre ${}_{\alpha}\longleftarrow : U \times A \longleftarrow U$ e ${}_{\beta}\longleftarrow : V \times A \longleftarrow V$ sse*

$$S \cdot \stackrel{a}{\longrightarrow}_{\alpha} \subseteq \stackrel{a}{\longrightarrow}_{\beta} \cdot S \ \wedge\ {}_{\beta}\stackrel{a}{\longleftarrow} \cdot S \subseteq S \cdot {}_{\alpha}\stackrel{a}{\longleftarrow} \tag{4.27}$$

para todo o $a \in A$.

Prova. A primeira parcela da conjunção define S como uma simulação. A segunda é derivada como se segue:

$$
\begin{array}{ll}
 & S^\circ \quad \text{é simulação} \\
\equiv & \quad \{ \text{ definição de simulação } \} \\
 & S^\circ \cdot \xrightarrow{a}_\beta \subseteq \xrightarrow{a}_\alpha \cdot S^\circ \\
\equiv & \quad \{ (\xrightarrow{a}_\gamma)^\circ = {}_\gamma\xleftarrow{a} \} \\
 & S^\circ \cdot ({}_\beta\xleftarrow{a})^\circ \subseteq ({}_\alpha\xleftarrow{a})^\circ \cdot S^\circ \\
\equiv & \quad \{ (R \cdot S)^\circ = S^\circ \cdot R^\circ \} \\
 & ({}_\beta\xleftarrow{a} \cdot S)^\circ \subseteq (S \cdot {}_\alpha\xleftarrow{a})^\circ \\
\equiv & \quad \{ \text{ monotonia: } R \subseteq S \equiv R^\circ \subseteq S^\circ \} \\
 & {}_\beta\xleftarrow{a} \cdot S \subseteq S \cdot {}_\alpha\xleftarrow{a}
\end{array}
$$

□

Prosseguindo o cálculo iniciado na prova deste resultado no sentido da introdução de variáveis, obtemos

$$
\begin{array}{ll}
 & {}_\beta\xleftarrow{a} \cdot S \subseteq S \cdot {}_\alpha\xleftarrow{a} \\
\equiv & \quad \{ \text{ conecção de Galois: } (R\cdot) \dashv (R\backslash) \} \\
 & S \subseteq {}_\beta\xleftarrow{a} \backslash (S \cdot {}_\alpha\xleftarrow{a}) \\
\equiv & \quad \{ \text{ introdução de variáveis } \} \\
 & \forall_{v\in V, u\in U} \,.\; v S u \Rightarrow v\,({}_\beta\xleftarrow{a} \backslash (S \cdot {}_\alpha\xleftarrow{a}))\,u \\
\equiv & \quad \{ \text{ definição de } \backslash \} \\
 & \forall_{v\in V, u\in U} \,.\; v S u \Rightarrow (\forall_{v'\in V} \,.\; v' \,{}_\alpha\xleftarrow{a}\, v \Rightarrow v'\,({}_\beta\xleftarrow{a} \cdot S)\,u') \\
\equiv & \quad \{ \text{ definição de } \cdot \} \\
 & \forall_{v\in V, u\in U} \,.\; v S u \Rightarrow (\forall_{v'\in V} \,.\; v' \,{}_\beta\xleftarrow{a}\, v \Rightarrow (\exists_{u'\in U} \,.\; u' \,{}_\alpha\xleftarrow{a}\, u \wedge v' S u'))
\end{array}
$$

expressão que, em conjunção com (4.25), é precisamente a utilizada nas apresentações clássicas de bissimulação (*cf.*, [Par81, Mil89]). Com esta formulação torna-se trivial verificar, para o caso particular que nos ocupa, a observação de que a existência de um morfismo entre duas coalgebras estabelece uma bissimulação entre os pares de estados por ele ligados. Formalmente,

Lema 18 *O grafo de um morfismo $h : \beta \longleftarrow \alpha$ entre coalgebras,* i.e.*, o próprio h visto como uma relação binária entre os espaços de estados de α e β, é uma bissimulação.*

Prova. Como vimos h é morfismo entre as coalgebras α e β sse satisfaz as inequações (4.19) e (4.20). A última equivale à segunda parcela da conjunção (4.27) que define bissimulação. Idêntica equivalência se verifica entre a primeira parcela dessa conjunção e (4.19):

$$
\begin{array}{ll}
 & h \cdot {}_{\alpha}\overleftarrow{}^{a} \subseteq {}_{\beta}\overleftarrow{}^{a} \cdot h \\
\equiv & \quad \{ \text{ lei (2.37)} \} \\
 & {}_{\alpha}\overleftarrow{}^{a} \subseteq h^{\circ} \cdot {}_{\beta}\overleftarrow{}^{a} \cdot h \\
\equiv & \quad \{ \text{ monotonia} \} \\
 & ({}_{\alpha}\overleftarrow{}^{a})^{\circ} \subseteq (h^{\circ} \cdot {}_{\beta}\overleftarrow{}^{a} \cdot h)^{\circ} \\
\equiv & \quad \{ \text{ converso} \} \\
 & \overrightarrow{}^{a}_{\alpha} \subseteq h^{\circ} \cdot \overrightarrow{}^{a}_{\beta} \cdot h \\
\equiv & \quad \{ \text{ lei (2.37)} \} \\
 & h \cdot \overrightarrow{}^{a}_{\alpha} \subseteq \overrightarrow{}^{a}_{\beta} \cdot h
\end{array}
$$

□

Lema 19 *O converso de uma bissimulação* $S : V \longleftarrow U$ *é ainda uma* bissimulação.

Prova.

$$
\begin{array}{ll}
 & S^{\circ} \text{ é bisimulação} \\
\equiv & \quad \{ \text{ definição de bisimulação} \} \\
 & S^{\circ} \cdot \overrightarrow{}^{a}_{\alpha} \subseteq \overrightarrow{}^{a}_{\beta} \cdot S^{\circ} \quad \wedge \quad {}_{\beta}\overleftarrow{}^{a} \cdot S^{\circ} \subseteq S^{\circ} \cdot {}_{\alpha}\overleftarrow{}^{a} \\
\equiv & \quad \{ (\overrightarrow{}^{a}_{\gamma})^{\circ} = {}_{\gamma}\overleftarrow{}^{a} \} \\
 & S^{\circ} \cdot ({}_{\alpha}\overleftarrow{}^{a})^{\circ} \subseteq ({}_{\beta}\overleftarrow{}^{a})^{\circ} \cdot S^{\circ} \quad \wedge \quad (\overrightarrow{}^{a}_{\beta})^{\circ} \cdot S^{\circ} \subseteq S^{\circ} \cdot (\overrightarrow{}^{a}_{\alpha})^{\circ} \\
\equiv & \quad \{ \text{ converso de uma composição} \} \\
 & ({}_{\alpha}\overleftarrow{}^{a} \cdot S)^{\circ} \subseteq (S \cdot {}_{\beta}\overleftarrow{}^{a})^{\circ} \quad \wedge \quad (S \cdot \overrightarrow{}^{a}_{\beta})^{\circ} \subseteq (\overrightarrow{}^{a}_{\alpha} \cdot S)^{\circ} \\
\equiv & \quad \{ \text{ monotonia} \} \\
 & {}_{\alpha}\overleftarrow{}^{a} \cdot S \subseteq S \cdot {}_{\beta}\overleftarrow{}^{a} \quad \wedge \quad S \cdot \overrightarrow{}^{a}_{\beta} \subseteq \overrightarrow{}^{a}_{\alpha} \cdot S \\
\equiv & \quad \{ \text{ hipótese} \} \\
 & \mathsf{true}
\end{array}
$$

□

Lema 20 *O conjunto de todas as (bi)simulações entre dois sistemas de transição forma um recticulado completo.*

Prova. Para o caso das simulações o resultado decorre directamente dos pontos (1) e (3) do lema 16 ($\bot$ é simulação e a união binária de simulações é ainda uma simulação). O argumento estende às bissimulações uma vez que estas são igualmente simulações.

□

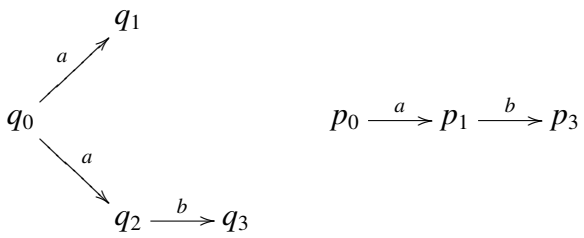
Definição 21 *Designamos por $\lesssim$ (respectivamente, $\sim$) a reunião de todas as simulações (respectivamente, bissimulações) do tipo adequado*[10]*,* i.e.*, os topos dos recticulados referidos no resultado anterior.*

Lema 21 *A relação $\lesssim$ é uma pré-ordem, enquanto $\sim$ é uma relação de equivalência.*

Prova. Por definição $\lesssim$ é a maior simulação. Logo, pelo lema 16, $\lesssim \cdot \lesssim \subseteq \lesssim$ e $\mathsf{id} \subseteq \sim$. Similarmente, $\sim \cdot \sim \subseteq \sim$ e $\mathsf{id} \subseteq \sim$. Adicionalmente, a simetria, $\sim^{\circ}=\sim$, vem da definição de $\sim$ como a maior bisimulação e do lema 16.

□

Note-se que $\lesssim$ *não* é uma ordem parcial. Atente-se no contra-exemplo seguinte, onde $q_0 \lesssim p_0$ e $p_0 \lesssim q_0$, mas, claramente, $p_0 \nsim q_0$:



4.6 Uma Caracterização da Equivalência Fraca

4.6.1 Bissimulação Fraca

Em álgebra de processos é usual definir noções de equivalência que abstraem da ocorrência de determinadas acções consideradas, num determinado nível de

[10]Em rigor, $\lesssim_{\alpha,\beta}$ e $\sim_{\alpha,\beta}$ de modo a referenciar estas noções às coalgebras entre as quais são definidas.

abstracção, *não observáveis* do exterior do sistema que se pretende descrever. Tipicamente estas acções etiquetam transições internas do sistema. Em CCS [Mil89], por exemplo, todas as transições internas, nomeadamente as que resultam da sincronização de acções complementares, são mapeadas na etiqueta τ.

Genericamente, consideremos um conjunto $\Upsilon \subseteq A$ de etiquetas correspondendo a acções (transições, efeitos) *não observáveis*. Com base em Υ podemos definir as noções de *simulação* e *bissimulação fraca* de modo análogo ao caso estrito apresentado anteriormente. É apenas necessário substituir na definição a família de relações de transição ${}_{\alpha}\overset{a}{\longleftarrow}$ por outra, que designaremos por ${}_{\alpha}\overset{a}{\Longleftarrow}$, que ignore a ocorrência de acções em Υ. Formalmente,

Definição 22 *Dados dois sistemas de transição* ${}_{\alpha}\longleftarrow : A \times U \longleftarrow U$ *e* ${}_{\beta}\longleftarrow : A \times V \longleftarrow V$ *definidos sobre um mesmo conjunto de etiquetas A e um subconjunto* $\Upsilon \subseteq A$ *de etiquetas não observáveis, uma* simulação fraca *de* ${}_{\alpha}\longleftarrow$ *em* ${}_{\beta}\longleftarrow$ *é uma relação* $S : V \longleftarrow U$ *tal que*

$$\begin{aligned}
\forall_{u\in U, v\in V}\,.\; v S u \;\Rightarrow\; & \\
& \forall_{a\in A-\Upsilon}\forall_{u'\in U}\,.\; u'\;{}_{\alpha}\overset{a}{\Longleftarrow}\; u \Rightarrow (\exists_{v'\in V}\,.\; v'\;{}_{\beta}\overset{a}{\Longleftarrow}\; v \wedge v' S u') \\
& \wedge \\
& \forall_{u'\in U}\,.\; u'\;{}_{\alpha}\Longleftarrow\; u \Rightarrow (\exists_{v'\in V}\,.\; v'\;{}_{\beta}\Longleftarrow\; v \wedge v' S u')
\end{aligned}$$

onde ${}_{\alpha}\Longleftarrow$ *é a união, para todo o* $\tau \in \Upsilon$*, do fecho transitivo e reflexivo de* ${}_{\alpha}\overset{\tau}{\longleftarrow}$*, que designamos por* $\mathsf{tr}({}_{\alpha}\overset{\tau}{\longleftarrow})$*. Por seu lado a relação* ${}_{\alpha}\overset{a}{\Longleftarrow}$ *é definida por abreviatura:*

$$ {}_{\alpha}\overset{a}{\Longleftarrow} \overset{\text{abv}}{=} {}_{\alpha}\Longleftarrow \cdot {}_{\alpha}\overset{a}{\longleftarrow} \cdot {}_{\alpha}\Longleftarrow \tag{4.28}$$

para todo o $a \in A - \Upsilon$*. Uma* bissimulação fraca *é uma simulação fraca cuja conversa é ainda uma simulação fraca.*

Claramente a reunião de todas as bissimulações fracas do tipo adequado é uma relação de equivalência que designaremos por $\approx$[11].

Este tipo de conceitos, em que se baseiam equivalências que têm um papel fundamental nas álgebras de processos, tem sido insuficientemente abordados a partir do ponto de vista coalgébrico. De facto, se, como vimos no lema 18, a existência de um morfismo entre coalgebras é suficiente para a identificação de uma bissimulação, não é fácil captar noções menos estritas de forma directa.

[11] Em rigor, $\approx_{\alpha,\beta}$ de modo a referenciar estas noções às coalgebras entre as quais são definidas.

A estratégia que vamos adoptar nesta dissertação para caracterizar coalgebricamente bissimulações fracas baseia-se numa transformação que a cada coalgebra α faz corresponder outra coalgebra $\widehat{\alpha}$ de tal forma que uma bissimulação estrita sobre $\widehat{\alpha}$ corresponda a uma bissimulação fraca sobre α. Deste modo a técnica usual de exibição de um morfismo para testemunhar uma bissimulação permanece válida, mas agora aplicada não sobre α, mas sobre $\widehat{\alpha}$.

A construção de $\widehat{\alpha}$ é ilustrada no diagrama seguinte, onde se recorre à transposição de relações, primeiro para obter o sistema de transição subjacente a α e, depois, para converter o novo sistema de transição na coalgebra $\widehat{\alpha}$.

$$\begin{array}{ccccc} \alpha : \mathcal{P}(A \times U) & \longleftarrow & U & \xrightarrow{\in \cdot} & {}_{\alpha}\longleftarrow : A \times U \longleftarrow U \\ & & & & \Big\downarrow \mathbb{O} \\ \widehat{\alpha} : \mathcal{P}(A \times U) & \longleftarrow & U & \xleftarrow{\Lambda} & {}_{\alpha}\leftsquigarrow : A \times U \longleftarrow U \end{array}$$

Assim

Definição 23 *Dada uma coalgebra $\alpha : \mathcal{P}(A \times U) \longleftarrow U$, define-se a correspondente* redução observacional *como a coalgebra*

$$\widehat{\alpha} \;=\; \Lambda\,\mathbb{O}\,(\in \cdot \alpha) \;=\; \Lambda\,\mathbb{O}\,({}_{\alpha}\longleftarrow) \;=\; \Lambda\ {}_{\alpha}\leftsquigarrow \tag{4.29}$$

onde ${}_{\alpha}\leftsquigarrow$ é definido da seguinte forma

$$(a, u')\ {}_{\alpha}\leftsquigarrow\ u \quad\equiv\quad u'\ {}_{\alpha}\overset{a}{\Longleftarrow}\ u \quad (\textit{se } a \notin \Upsilon) \tag{4.30}$$

$$(\tau, u')\ {}_{\alpha}\leftsquigarrow\ u \quad\equiv\quad u'\ {}_{\alpha}\Longleftarrow\ u \quad (\textit{se } \tau \in \Upsilon) \tag{4.31}$$

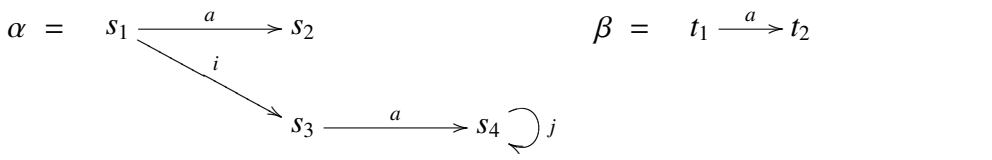
É imediato verificar que a noção de bissimulação fraca sobre duas coalgebras coincide com a de bissimulação estrita entre as respectivas reduções observacionais.

Lema 22 *Dadas duas coalgebras $\alpha : \mathcal{P}(Act \times U) \longleftarrow U$ e $\beta : \mathcal{P}(Act \times V) \longleftarrow V$, se existir um morfismo $h : \widehat{\beta} \longleftarrow \widehat{\alpha}$ tal que $v = h\,u$, então $v \approx_{\alpha,\beta} u$.*

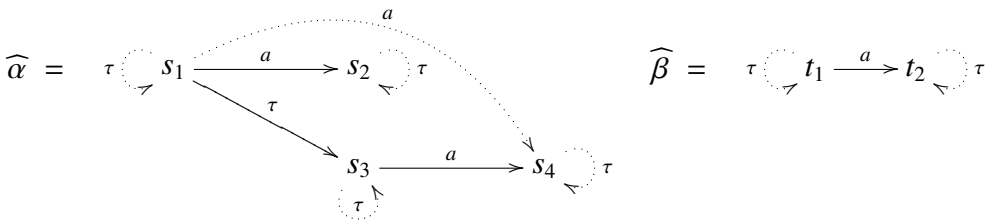
Prova. Pelo lema 18 tem-se $v \approx_{\widehat{\alpha},\widehat{\beta}} u$. No entanto, de acordo com a definição 23, as transições associadas a $\widehat{\alpha}$ e $\widehat{\beta}$ coincidem com as transições observáveis dos sistemas de transição associados a α e β, respectivamente, o que estabelece o resultado.

□

Exemplo. Comecemos por considerar as coalgebras α e β representadas pelos seguintes sistemas de transição:



Suponhamos ainda que $\Upsilon = \{i, j\}$ é dado como o conjunto das acções não observáveis. As respectivas reduções observacionais são



É agora possível definir um morfismo $h : \{t_1, t_2\} \longleftarrow \{s_1, s_2, s_3, s_4\}$ que, entre outros, liga os estados s_1 e t_1, *i.e.*, os estados iniciais dos autómatos codificados por $\widehat{\alpha}$ e $\widehat{\beta}$, respectivamente. Assim,

$$\begin{aligned} h\, s_1 &= h\, s_3 = t_1 \\ h\, s_2 &= h\, s_4 = t_2 \end{aligned}$$

Claramente $\widehat{\beta} \cdot h = \mathcal{P}(\mathsf{id} \times h) \cdot \widehat{\alpha}$. De facto,

$$\begin{aligned} (\widehat{\beta} \cdot h)\, s_1 = (\widehat{\beta} \cdot h)\, s_3 &= \widehat{\beta}\, t_1 = \{(\tau, t_1), (a, t_2)\} \\ (\mathcal{P}(\mathsf{id} \times h) \cdot \widehat{\alpha})\, s_1 &= \mathcal{P}(\mathsf{id} \times h)\, \{(\tau, s_1), (\tau, s_3), (a, s_2), (a, s_4)\} = \{(\tau, t_1), (a, t_2)\} \\ (\mathcal{P}(\mathsf{id} \times h) \cdot \widehat{\alpha})\, s_3 &= \mathcal{P}(\mathsf{id} \times h)\, \{(\tau, s_3), (a, s_4)\} = \{(\tau, t_1), (a, t_2)\} \end{aligned}$$

e

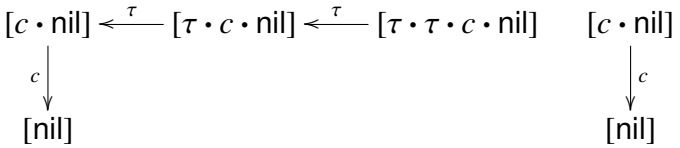
$$\begin{aligned} (\widehat{\beta} \cdot h)\, s_2 = (\widehat{\beta} \cdot h)\, s_4 &= \widehat{\beta}\, t_1 = \{(\tau, t_2)\} \\ (\mathcal{P}(\mathsf{id} \times h) \cdot \widehat{\alpha})\, s_2 &= \mathcal{P}(\mathsf{id} \times h)\, \{(\tau, s_2)\} = \{(\tau, t_2)\} \\ (\mathcal{P}(\mathsf{id} \times h) \cdot \widehat{\alpha})\, s_3 &= \mathcal{P}(\mathsf{id} \times h)\, \{(\tau, s_4)\} = \{(\tau, t_2)\} \end{aligned}$$

Assim $s_1 \sim_{\widehat{\alpha},\widehat{\beta}} t_1$ e, logo, $s_1 \approx_{\alpha,\beta} t_1$.

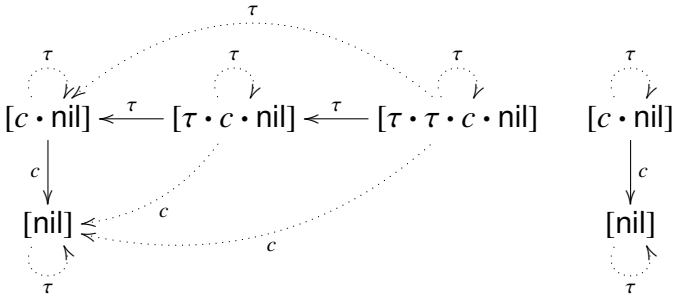
✓

Exemplo. Consideremos, agora, os processos $c \cdot P$ e $\tau \cdot \tau \cdot c \cdot P$ em CCS, onde, por convenção, τ representa a acção não observável. Recorde-se que $\omega : \mathcal{P}(Act \times \nu) \longleftarrow \nu$, a coalgebra final para o functor $\mathcal{P}(Act \times \mathsf{Id})$, representa o universo das denotações dos processos. Como já referimos, o seu portador, ν, contém as classes de equivalência para $\sim$ de árvores de ramificação finita etiquetadas por acções em Act. Para cada termo, ou processo, P, representamos por $[P] \in \nu$ a sua denotação. Suponhamos que pretendemos mostrar que os dois processos acima são observacionalmente equivalentes.

No diagrama seguinte representam-se os fragmentos de ω relevantes:



Calculemos agora os fragmentos correspondentes na redução observacional da coalgebra final ω, *i.e.*, em $\widehat{\omega}$:



O morfismo $h : \widehat{\omega} \longleftarrow \widehat{\omega}$ definido por

$$h\,[\tau \cdot \tau \cdot c \cdot \mathsf{nil}] \;=\; h\,[\tau \cdot c \cdot \mathsf{nil}] \;=\; [c \cdot \mathsf{nil}]$$

e pela identidade em todos os restantes elementos de ν, estabelece uma bisimulação estrita sobre $\widehat{\omega}$ entre as denotações dos processos $c \cdot \mathsf{nil}$ e $\tau \cdot \tau \cdot c \cdot \mathsf{nil}$ e, logo, o resultado desejado.

✓

Como seria de esperar, tem-se

$$\sim \;\subseteq\; \approx \tag{4.32}$$

conforme se prova no seguinte lema.

Lema 23 *Qualquer morfismo h de $\alpha : \mathcal{P}(Act \times U) \longleftarrow U$ para $\beta : \mathcal{P}(Act \times V) \longleftarrow V$ é, igualmente, um morfismo de $\widehat{\alpha}$ para $\widehat{\beta}$.*

Prova. Consideremos um h tal que $\beta \cdot h = \mathcal{P}(\mathsf{id} \times h) \cdot \alpha$ e mostremos que

$$\widehat{\beta} \cdot h = \mathcal{P}(\mathsf{id} \times h) \cdot \widehat{\alpha} \tag{4.33}$$

$$\begin{array}{ll}
 & \widehat{\beta} \cdot h \;=\; \mathcal{P}(\mathsf{id} \times h) \cdot \widehat{\alpha} \\
\equiv & \quad \{ \text{ definição de redução observável } \} \\
 & \Lambda \cdot {}_{\beta}\!\leftsquigarrow \cdot h \;=\; \mathcal{P}(\mathsf{id} \times h) \cdot \Lambda \cdot {}_{\alpha}\!\leftsquigarrow \\
\equiv & \quad \{ \; \Lambda(f \cdot R) = \mathcal{P} f \cdot \Lambda R \; \} \\
 & \Lambda \cdot {}_{\beta}\!\leftsquigarrow \cdot h \;=\; \Lambda \cdot (\mathsf{id} \times h) \cdot {}_{\alpha}\!\leftsquigarrow \\
\equiv & \quad \{ \Lambda \text{ é um isomorfismo } \} \\
 & {}_{\beta}\!\leftsquigarrow \cdot h \;=\; (\mathsf{id} \times h) \cdot {}_{\alpha}\!\leftsquigarrow
\end{array}$$

Para provar esta igualdade fixemos um $a \notin \Upsilon$, arbitrário, e verifiquemos que ${}_{\beta}\!\overset{a}{\leftsquigarrow} \cdot h \;=\; h \cdot {}_{\alpha}\!\overset{a}{\leftsquigarrow}$. Então,

$$\begin{array}{ll}
 & {}_{\beta}\!\overset{a}{\leftsquigarrow} \cdot h \\
\equiv & \quad \{ \text{ pela definição de } {}_{\beta}\!\leftsquigarrow \} \\
 & \beta \Longleftarrow \cdot \beta \overset{a}{\longleftarrow} \cdot \beta \Longleftarrow \cdot h \\
\equiv & \quad \{ \; \beta \Longleftarrow = \mathsf{tr}(\beta \overset{\tau}{\longleftarrow}) \; \} \\
 & \mathsf{tr}(\beta \overset{\tau}{\longleftarrow}) \cdot \beta \overset{a}{\longleftarrow} \cdot \mathsf{tr}(\beta \overset{\tau}{\longleftarrow}) \cdot h \\
\equiv & \quad \{ \text{ Lema 14 } \} \\
 & \mathsf{tr}(\beta \overset{\tau}{\longleftarrow}) \cdot \beta \overset{a}{\longleftarrow} \cdot h \cdot \mathsf{tr}(\alpha \overset{\tau}{\longleftarrow}) \\
\equiv & \quad \{ \; h : \beta \longleftarrow \alpha \text{ é morfismo } \} \\
 & \mathsf{tr}(\beta \overset{\tau}{\longleftarrow}) \cdot h \cdot \alpha \overset{a}{\longleftarrow} \cdot \mathsf{tr}(\alpha \overset{\tau}{\longleftarrow}) \\
\equiv & \quad \{ \text{ Lema 14 } \} \\
 & h \cdot \mathsf{tr}(\alpha \overset{\tau}{\longleftarrow}) \cdot \alpha \overset{a}{\longleftarrow} \cdot \mathsf{tr}(\alpha \overset{\tau}{\longleftarrow}) \\
\equiv & \quad \{ \; \alpha \Longleftarrow = \mathsf{tr}(\alpha \overset{\tau}{\longleftarrow}) \; \} \\
 & h \cdot \alpha \Longleftarrow \cdot \alpha \overset{a}{\longleftarrow} \cdot \alpha \Longleftarrow \\
\equiv & \quad \{ \text{ pela definição de } {}_{\beta}\!\leftsquigarrow \} \\
 & h \cdot {}_{\alpha}\!\overset{a}{\leftsquigarrow}
\end{array}$$

□

Uma instância da inclusão 4.32 é bem conhecida em álgebra de processos, nomeadamente no desenvolvimento das caracterizações observacionais de processos em CCS [Mil99]. Note-se, porém, que a sua prova no lema 23, é mais geral: aplica-se a qualquer coalgebra para o functor que temos vindo a considerar e não apenas à respectiva coalgebra final.

4.6.2 Exemplos

Investiguemos agora a utilização desta abordagem à bissimulação fraca para discutir alguns exemplos de equivalência observacional em CCS, onde $\Upsilon = \{\tau\}$. Claramente a coalgebra de trabalho é a coalgebra final $\omega : \mathcal{P}(Act \times \nu) \longleftarrow \nu$.

Exemplo. Comecemos por considerar o resultado que em CCS caracteriza a equivalência observacional:

$$p \approx \tau \cdot p \tag{4.34}$$

De acordo com a discussão anterior, torna-se necessário encontrar um morfismo entre as reduções observacionais de ω, *i.e.*, $h : \widehat{\omega} \longleftarrow \widehat{\omega}$, que relacione os dois processos. Vamos definir

$$\begin{aligned} h(\tau \cdot p) &= p \quad \text{para todo o } p \in \nu \\ h &= \mathsf{id} \quad \text{nos restantes casos} \end{aligned}$$

e mostrar que h, assim definido, é de facto um morfismo, *i.e.*,

$$\widehat{\omega} \cdot h = \mathcal{P}(\mathsf{id} \times h) \cdot \widehat{\omega} \tag{4.35}$$

Dada a definição de h, a equação (4.35) será válida se se verificar $\widehat{\omega}(\tau \cdot p) = \widehat{\omega}p$, o que se prova por

$$\begin{array}{ll} & \widehat{\omega}(\tau \cdot p) \\ \equiv & \quad \{ \text{ definição de redução observacional } \} \\ & \Lambda\, {}_{\omega}\!\leftsquigarrow\, (\tau \cdot p) \\ \equiv & \quad \{ \text{ definição de } {}_{\omega}\!\leftsquigarrow \text{ e transposição } \} \\ & \{(a, p') |\ p'\ {}_{\omega}\!\overset{a}{\Longleftarrow}\ \tau \cdot p\} \cup \{(\tau, p') |\ p'\ {}_{\omega}\!\Longleftarrow\ \tau \cdot p\} \\ \equiv & \quad \{ \text{ definição de } {}_{\omega}\!\Longleftarrow \} \\ & \{(a, p') |\ p'\ {}_{\omega}\!\overset{a}{\Longleftarrow}\ p\} \cup \{(\tau, p') |\ p'\ {}_{\omega}\!\Longleftarrow\ p\} \\ \equiv & \quad \{ \text{ definição de } {}_{\omega}\!\leftsquigarrow \text{ e transposição } \} \end{array}$$

$$\begin{array}{ll} & \Lambda\ {}_{\omega}\!\!\leftsquigarrow\ p \\ \equiv & \quad \{\ \text{definição de redução observacional}\ \} \\ & \widehat{\omega}\, p \end{array}$$

✓

Exemplo. Tentemos, agora, tratar outro tipo de leis cuja formulação envolve hipóteses adicionais sobre os processos. Comecemos por discutir a validade da seguinte propriedade

$$p + q \ \approx\ p' + q' \quad \Leftarrow \quad p' \approx p \ \wedge\ q' \approx q \tag{4.36}$$

teremos, de novo, de definir um h relacionando os termos relevantes e mostrar que tal h satisfaz as condições de morfismo, *i.e.*

$$\widehat{\omega} \cdot + \cdot (h \times h) \ =\ \mathcal{P}(\mathsf{id} \times h) \cdot \widehat{\omega} \cdot + \tag{4.37}$$

equação que pressupõe uma definição do combinador + sobre $\widehat{\omega}$. Recorde-se que $\omega \cdot + \ = \cup \cdot (\omega \times \omega)$. Combinando esta definição com a de redução observacional em (4.29), vem

$$\widehat{\omega}([p + q]) \ =\ \widehat{\omega} p \ \cup \widehat{\omega} q \ \cup\ \{(\tau, r) |\ r\ {}_{\omega}\!\!\Longleftarrow\ [p + q]\} \tag{4.38}$$

Assim,

$$\widehat{\omega}\,([a \cdot \mathsf{nil} \ +\ b \cdot \mathsf{nil}]) \ =\ \{(a, [\mathsf{nil}]), (b, [\mathsf{nil}]), (\tau, [(a \cdot \mathsf{nil} \ +\ b \cdot \mathsf{nil})])\}$$

enquanto

$$\widehat{\omega}\,([\tau \cdot a \cdot \mathsf{nil} \ +\ b \cdot \mathsf{nil}]) \ =\ \{(a, [\mathsf{nil}]), (b, [\mathsf{nil}]), (\tau, [(\tau \cdot a \cdot \mathsf{nil} \ +\ b \cdot \mathsf{nil})]), (\tau, [a \cdot \mathsf{nil}])\}$$

apesar de, como se mostrou no exemplo anterior, $a \cdot \mathsf{nil} \approx \tau \cdot a \cdot \mathsf{nil}$, pelo que (4.36) não se verifica.

✓

Na reconstrução coindutiva das álgebras de processos que temos vindo a discutir neste capítulo, leis como (4.36), que traduzem a preservação de operadores

são automaticamente assumidas quando a relação em causa é a equivalência comportamental, *i.e.*, a maior das bissimulações estritas que representamos por $\sim$. De facto, o nosso objecto de estudo não são termos sintácticos (como, *e.g.*, em [Mil80] onde este tipo de leis são explicitamente formuladas) mas as suas denotações numa coalgebra final ω. Ora em qualquer coalgebra final $\sim$ coincide com a igualdade, pelo que todas estas leis são trivialmente verdadeiras. De facto, só faz sentido discutir este tipo de leis em coalgebras não finais — por exemplo, em coalgebras cujos portadores fossem conjuntos de termos definidos por uma assinatura.

Como se viu no exemplo anterior, porém, este tipo de leis deve ser investigado quando formuladas em termos de relações menos finas que a igualdade, por exemplo, em termos de $\approx$. Esse é, ainda, o caso do próximo exemplo.

Exemplo. Consideremos, por fim, uma lei de congruência, mas envolvendo um combinador estático, logo, definido recursivamente.

$$p \mid\mid\mid q \;\approx\; p' \mid\mid\mid q' \quad \Leftarrow \quad p' \approx p \;\wedge\; q' \approx q \tag{4.39}$$

Suponhamos que as equivalências $p' \approx p$ e $q' \approx q$ são testemunhadas por morfismos $f : \widehat{\omega} \longleftarrow \widehat{\omega}$ e $g : \widehat{\omega} \longleftarrow \widehat{\omega}$, não necessariamente coincidentes. Definamos uma função $h : \nu \longleftarrow \nu$ tal que

$$h \cdot \mid\mid\mid \;=\; \mid\mid\mid \cdot (f \times g) \tag{4.40}$$

comportando-se como a identidade em todos os outros casos. Mostremos que h assim definido é, de facto, um morfismo entre as reduções observacionais de ω, *i.e.*,

$$\widehat{\omega} \cdot h \;=\; \mathcal{P}(\mathsf{id} \times h) \cdot \widehat{\omega} \tag{4.41}$$

Claramente, a equação (4.41) é trivialmente verdadeira para todos os argumentos em que h se comporta como a identidade. Assim, o único caso que carece de verificação é aquele em que o argumento é uma intercalação. Mostremos, então, que

$$\widehat{\omega} \cdot h \cdot \mid\mid\mid \;=\; \mathcal{P}(\mathsf{id} \times h) \cdot \widehat{\omega} \cdot \mid\mid\mid \tag{4.42}$$

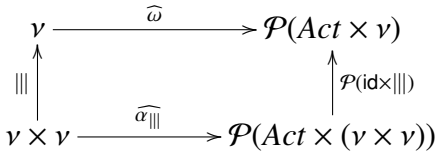
Por definição de h,

$$\widehat{\omega} \cdot h \cdot \mid\mid\mid \;=\; \widehat{\omega} \cdot \mid\mid\mid \cdot (f \times g)$$

Mas a que pode a expressão $\widehat{\omega} \cdot \mid\mid\mid$ ser igualada? Repare-se que o diagrama

$$\begin{array}{ccc} \nu & \xrightarrow{\;\omega\;} & \mathcal{P}(Act \times \nu) \\ {\scriptstyle \mid\mid\mid}\uparrow & & \uparrow{\scriptstyle \mathcal{P}(\mathsf{id} \times \mid\mid\mid)} \\ \nu \times \nu & \xrightarrow{\;\alpha_{\mid\mid\mid}\;} & \mathcal{P}(Act \times (\nu \times \nu)) \end{array}$$

generaliza a



onde

$$\widehat{\alpha_{|||}} \;=\; \Lambda\ {}_{\alpha_{|||}} \leftsquigarrow \tag{4.43}$$

do mesmo modo que $\widehat{\omega} \;=\; \Lambda\ {}_{\omega} \leftsquigarrow$. Na verdade, pelo lema 23, o último diagrama é implicado pelo primeiro. Não é difícil encontrar uma definição directa para $\Lambda\ {}_{\alpha_{|||}} \leftsquigarrow$: em rigor, basta substituir ω por $\widehat{\omega}$ na definição de $\alpha_{|||}$ registada no início deste capítulo e considerar ainda uma transição etiquetada por τ do par de processos argumentos para ele próprio. Um exemplo ajudará a clarificar esta definição: $\alpha_{|||}\,(\tau \cdot a \cdot \mathsf{nil}, b \cdot \mathsf{nil})$ será o conjunto

$$\{(\tau, ([a \cdot \mathsf{nil}], [b \cdot \mathsf{nil}])), (a, ([\mathsf{nil}], [b \cdot \mathsf{nil}])), (b, ([a \cdot \mathsf{nil}], [\mathsf{nil}]))\}$$

reunido com o conjunto singular

$$\{(\tau, ([\tau \cdot a \cdot \mathsf{nil}], [b \cdot \mathsf{nil}]))\}$$

Assim,

$$\widehat{\alpha_{|||}} \;=\; \cup \cdot (\alpha_1 \times \alpha_2) \cdot \triangle \tag{4.44}$$

onde

$$\begin{aligned} \alpha_1 \;&=\; \cup \cdot (\tau_r \times \tau_l) \cdot ((\widehat{\omega} \times \mathsf{id}) \times (\mathsf{id} \times \widehat{\omega})) \cdot \triangle \\ \alpha_1 \;&=\; \mathsf{sing} \cdot \mathsf{label}_\tau \end{aligned}$$

Verifiquemos, agora, que

$$\mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_1 \cdot (f \times g) \;=\; \mathcal{P}(\mathsf{id} \times h) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_1 \tag{4.45}$$

e, similarmente,

$$\mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_2 \cdot (f \times g) \;=\; \mathcal{P}(\mathsf{id} \times h) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_2 \tag{4.46}$$

A igualdade (4.45) é estabelecida por

$$
\begin{array}{ll}
& \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_1 \cdot (f \times g) \\
= & \quad \{ \text{ definição de } \alpha_1 \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\widehat{\omega} \times \mathsf{id}) \times (\mathsf{id} \times \widehat{\omega})) \cdot \triangle \cdot (f \times g) \\
= & \quad \{ \triangle \text{ natural, } \times \text{ functor} \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot (((\widehat{\omega} \cdot f) \times g) \times (f \times (\widehat{\omega} \cdot g))) \cdot \triangle \\
= & \quad \{ f \text{ e } g \text{ são morfismos, } \times \text{ functor} \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot (((\mathcal{P}(\mathsf{id} \times f) \times g) \times (f \times \mathcal{P}(\mathsf{id} \times g))) \cdot ((\widehat{\omega} \times \mathsf{id}) \times (\mathsf{id} \times \widehat{\omega})) \cdot \triangle \\
= & \quad \{ \tau_r \text{ e } \tau_l \text{ naturais} \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\mathcal{P}(\mathsf{id} \times (f \times g)) \times \mathcal{P}(\mathsf{id} \times (f \times g))) \cdot (\tau_r \times \tau_l) \cdot ((\widehat{\omega} \times \mathsf{id}) \times (\mathsf{id} \times \widehat{\omega})) \cdot \triangle \\
= & \quad \{ \cup \text{ natural} \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \mathcal{P}(\mathsf{id} \times (f \times g)) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\widehat{\omega} \times \mathsf{id}) \times (\mathsf{id} \times \widehat{\omega})) \cdot \triangle \\
= & \quad \{ \text{ definição de } \alpha_1 \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \mathcal{P}(\mathsf{id} \times (f \times g)) \cdot \alpha_1 \\
= & \quad \{ (4.40) \} \\
& \mathcal{P}(\mathsf{id} \times h) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_1
\end{array}
$$

Por outro lado a igualdade (4.46) é verificada pelo cálculo seguinte:

$$
\begin{array}{ll}
& \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_2 \cdot (f \times g) \\
= & \quad \{ \text{ definição de } \alpha_2 \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \mathsf{sing} \cdot \mathsf{label}_\tau \cdot (f \times g) \\
= & \quad \{ \text{ definição de sing e } \mathsf{label}_\tau \} \\
& \mathsf{sing} \cdot \mathsf{label}_\tau \cdot ||| \cdot (f \times g) \\
= & \quad \{ (4.40) \} \\
& \mathsf{sing} \cdot \mathsf{label}_\tau \cdot h \cdot ||| \\
= & \quad \{ \text{ definição de sing, } \mathsf{label}_\tau \text{ e } \alpha_2 \} \\
& \mathcal{P}(\mathsf{id} \times h) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_2
\end{array}
$$

Estamos, agora, em condições de verificar a equação (4.42). Assim,

$$
\begin{array}{ll}
& \widehat{\omega} \cdot h \cdot ||| \\
= & \quad \{ (4.40) \} \\
& \widehat{\omega} \cdot ||| \cdot (f \times g) \\
= & \quad \{ ||| \text{ é morfismo para } \widehat{\omega} \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \widehat{\alpha_{|||}} \cdot (f \times g)
\end{array}
$$

$$
\begin{aligned}
= & \quad \{ \text{ definição de } \widehat{\alpha_{|||}} \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\alpha_1 \times \alpha_2) \cdot \triangle \cdot (f \times g) \\
= & \quad \{ \triangle \text{ natural} \} \\
& \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\alpha_1 \times \alpha_2) \cdot ((f \times g) \times (f \times g)) \cdot \triangle \\
= & \quad \{ \cup \text{ natural}, \times \text{ functor} \} \\
& \cup \cdot ((\mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_1 \cdot (f \times g)) \times (\mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_2 \cdot (f \times g))) \cdot \triangle \\
= & \quad \{ (4.45) \text{ e } (4.46) \} \\
& \cup \cdot ((\mathcal{P}(\mathsf{id} \times h) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_1) \times (\mathcal{P}(\mathsf{id} \times h) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \alpha_2)) \cdot \triangle \\
= & \quad \{ \cup \text{ natural} \} \\
& \mathcal{P}(\mathsf{id} \times h) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \cup \cdot (\alpha_1 \times \alpha_2) \cdot \triangle \\
= & \quad \{ \text{ definição de } \widehat{\alpha_{|||}} \} \\
& \mathcal{P}(\mathsf{id} \times h) \cdot \mathcal{P}(\mathsf{id} \times |||) \cdot \widehat{\alpha_{|||}} \\
= & \quad \{ ||| \text{ é morfismo para } \widehat{\omega} \} \\
& \mathcal{P}(\mathsf{id} \times h) \cdot \widehat{\omega} \cdot |||
\end{aligned}
$$

✓

Capítulo 5

Conclusões e Trabalho Futuro

Resumo

Este capítulo conclui a dissertação, resumindo o que foi feito e apontando algumas pistas para trabalho futuro.

A Álgebra é a "Sciência que tem por objecto ensinar os meios de reduzir a regras gerais a resolução de todas as questões que se podem propor em termo de quantidades"
(in *Elementos de Análise (2 vols.), 1801 - 3ª ed. - 1º volume*)

Enquanto propriedades universais, as coalgebras finais fornecem descrições abstractas de estruturas comportamentais. A finalidade pode ser vista como uma base para o desenvolvimento do cálculo de programas coindutivos, quer como método de definição, quer como princípio de prova. O objectivo geral desta dissertação foi estudar a sua aplicação na programação coindutiva, ao nível da definição e da prova. Além disso, implementámos alguns combinadores e construímos funções em HASKELL. O trabalho foi organizado em dois casos de estudo.

No primeiro caso de estudo, estudamos as sequências infinitas que, sendo tipos coindutivos, podem ser observadas pela dinâmica da coalgebra final.A principal contribuição é a comparação entre um estilo de definição e prova a que chamamos *coindução explícita* (usado, nomeadamente, em [Rut05]) e a *coindução implícita*, ou por *cálculo* baseada nas propriedades universais dos combinadores anamorfismo, apomorfismo e futumorfismo. Adicionalmente foi construída uma biblioteca em HASKELL.

Comparando os dois estilos acima citados, *i.e.*, *coindução explícita* e *coindução implícita*, podemos dizer que este último é facilmente generalizado a outros casos de tipos coindutivos. Por outro lado, parece-nos ser mais adequado ao cálculo, sendo o seu esquema de prova e definição mais fácil de identificar e, portanto, também mais fácil de reutilizar noutros contextos. Em particular, este estilo parece, por isso, mais fácil de ensinar e de aprender. Pelas mesmas razões este estilo é mais facilmente susceptível de ser automatizado. Além disso, segue a prática da programação funcional.

O segundo caso de estudo incidiu na especificação coindutiva da álgebra de processos e sua prototipagem e visualização em HASKELL. A especificação é paramétrica na álgebra de sincronização. Foram implementadas as álgebras do CCS CSP e uma outra álgebra de sincronização a que chamamos co-ocorrência. Definimos alguns novos operadores e exploraram-se estilos de prova por cálculo. De notar que alguns novos operadores foram definidos como apomorfismos. Assim, ao tentar descobrir algumas das suas propriedades, necessitámos de desenvolver uma lei de fusão condicional para o apomorfismo.

Outra importante contribuição diz respeito à caracterização coalgébrica da bissimulação fraca. Para isso foi necessário explorar a transposição entre os níveis coalgébrico e relacional. De passagem diversos resultados clássicos sobre simulações e bissimulações foram provados de forma equacional e *pointfree* recorrendo ao cálculo das relações binárias. De novo observamos o carácter conciso e elegante das provas obtidas, quando comparadas com a literatura clássica em álgebra de processos. A caracterização da bissimulação fraca emerge desse esforço e parece ser mais simples e estrutural, embora porventura menos genérica, que aproximações mais abstractas, baseadas na acessibilidade sintáctica de certos functores, descritas em [RM02] e [SVW05]. Determinar até que ponto a nossa abordagem generaliza a coalgebras para classes mais amplas de functores, é um dos pontos que indicamos para trabalho futuro.

5.1 Trabalho Futuro

Como trabalho futuro há a destacar três áreas distintas. A primeira diz respeito aos esquemas de definição e prova coindutiva; a segunda à construção de uma interface gráfica para o animador de cálculos de processos; e, *the last but not the least*, a terceira concerne à generalização da aproximação aqui proposta à bissimulação fraca. Vejamos cada uma por si própria.

1. As técnicas de definição e prova coindutiva associadas a anamorfismos, apomorfismos ou mesmo à generalização destes últimos em futumorfismos, não

cobrem todo o espectro de problemas que seria desejável especificar e manipular coindutivamente. Um exemplo surge com a seguinte definição de um produto tensorial sobre A^ω, dada em [Rut05]:

$$\sigma \times \tau = (\sigma_0 \cdot \tau_0, (\sigma_0 \cdot \tau_1) + (\sigma_1 \cdot \tau_0), (\sigma_0 \cdot \tau_2) + (\sigma_1 \cdot \tau_1) + (\sigma_2 \cdot \tau_0), \ldots)$$

ou seja, o termo de ordem n corresponde a

$$(\sigma \times \tau)(n) = \sum_{k=0}^{n} \sigma_k \cdot \tau_{n-k}$$

Claramente esta função não cai em nenhuma das famílias estudadas nesta dissertação, na medida em que a definição exige um determinado tipo de processamento *após* realizada a invocação recursiva da função.

O mesmo se passa, de resto, na visão mais clássica a que chamamos *coindução explícita*: em muitos casos a construção de bissimulações envolve relações muito grandes e está longe de ser trivial. Esse facto tem motivado a proposta de *bissimulações a menos de* (ver, por exemplo, [Mil89]). A procura de versões progressivamente mais gerais dos princípios coinduticos tem conduzido, recentemente, à formulação de esquemas baseados em *bialgebras* [Bar01c]. O seu desenvolvimento no estilo calculacional adoptado nesta dissertação permanece em aberto.

2. Ao nível do cálculo de processos, seria desejável desenvolver um interface gráfico para o animador construído neste trabalho. A função `execProc`, já desenvolvida e documentada em Apêndice, é exemplo do tipo de ferramentas necessárias.

   Pode-se, ainda, continuar o desenvolvimento de novos operadores com base no futumorfismo, caracterizar as suas propriedades e provar uma lei de fusão condicional para o futumorfismo, que será certamente semelhante à do anamorfismo e do apomorfismo.

3. Referimos já o desafio da generalização da aproximação proposta aqui para caracterizar a bissimulação fraca a outros functores. Naturalmente, esses functores deverão ter pelo menos um *atributo* que represente alguma forma de acção internalizável. No cálculo de processos discutido no capítulo 4, tratamos estes como habitantes da coalgebra final para o functor $\mathcal{P}(Act \times \mathsf{Id})$.

O tratamento generaliza de forma imediata ao caso dos processos determinísticos. O functor é, nesse caso, $Act \times \mathsf{Id}$ e a coalgebra final é

$$\omega \;=\; \langle \mathsf{hd}, \mathsf{tl} \rangle : Act \times Act^{\omega} \longleftarrow Act$$

Da mesma forma, irá induzir uma relação, de transição observável que aglutinará num passo etiquetado por τ todos os passos correspondentes a elementos da stream que se considerem internos.

A dificuldade de tornar esta abordagem mais geral reside no papel essencial que nela tem a transposição Λ entre este tipo de coalgebras e as relações binárias. No caso das streams, existe uma transposição similar entre coalgebras para $Act \times \mathsf{Id}$, que são funções parciais, e, de novo as relações. Para outras famílias de functores e coalgebras haverá transposições interessantes? Isto é, processos de mapeamento por isomorfismo natural para domínios que sejam mais ricos ou adequados do ponto de vista do poder de cálculo?

Por fim, o animador de processos deverá ser estendido no sentido de cobrir a animação observacional de processos, *i.e.*, em termos da relação de transição observável. Trabalho preliminar começou já a ser feito nesse sentido.

Apêndice A

Coindução nas Streams

```
module Corecursion where

import BasicLib
import Functors

-- (1) Tipos como functores ---

newtype Mu f = In (f (Mu f))

unIn :: Mu f -> f (Mu f)
unIn (In x) = x

newtype Nu f = Fin (f (Nu f))

unFin :: Nu f -> f (Nu f)
unFin (Fin x) = x

-- Funções auxiliares ---

join :: (a -> c) -> (b -> c) -> Either a b -> c
join f g (Left x) = f x
join f g (Right y) = g y

lastF :: c -> Mu (EitherF f c)
lastF x = In (EitherF (i1 x))

consF :: f (Mu (EitherF f c)) -> Mu (EitherF f c)
```

```
consF x = In (EitherF (i2 x))

joinF :: (a -> c) -> (f b -> c) -> EitherF f a b -> c
joinF f g (EitherF x) = join f g x

-- (2) Esquemas recursivos ---

ana :: Functor f => (c -> f c) -> c -> Nu f
ana phi = Fin . fmap (ana phi) . phi

apo :: Functor f => (c -> f (Either c (Nu f))) -> c -> Nu f
apo phi = Fin . fmap (either (apo phi) id) . phi

futu :: Functor f => (c -> f (Mu (EitherF f c))) -> c -> Nu f
futu phi = ana (joinF phi id . unIn) . lastF

-- (3) Streams ---

type Stream a = Nu (S a)

-- Observadores na coalgebra final

headS :: Stream a -> a
headS s = case unFin s of St x _ -> x

tailS :: Stream a -> Stream a
tailS s = case unFin s of St _ r -> r

-- (4) Exemplos de Anamorfismos ---

iterateS :: (a -> a) -> a -> Stream a
iterateS f = ana phi
             where phi x = St x (f x)

mapS :: (a -> a) -> Stream a -> Stream a
mapS f =ana phi
        where phi x = St (f (headS x)) (tailS x)

natS = iterateS succ 1

oneS = iterateS (const 1) 1

odd :: Stream a -> Stream a
```

```
odd = ana phi
      where phi a = St ((headS . tailS) a) ((tailS . tailS) a)

even :: Stream a -> Stream a
even = ana phi
       where phi a = St (headS a) ((tailS . tailS) a)

zipS :: (Stream a, Stream b) -> Stream (a,b)
zipS = ana phi
       where phi (aa, bb) = St ((headS >< headS) (aa, bb))
                               ((tailS >< tailS) (aa, bb))

concatS :: (Stream a, Stream a) -> Stream a
concatS = ana phi
          where phi (aa, bb) = St (headS aa) (swap ((tailS aa), bb))

calldupS :: Stream a -> Stream a
calldupS = dupS . (split id (const True))

dupS :: (Stream a, Bool) -> Stream a
dupS = ana phi
       where phi (aa, b) = St (headS aa)
                              (cond (const b)
                              (split p1 (not . p2))
                              (split (tailS . p1) (not . p2)) (aa,b))

-- (5) Exemples de Apomorfismos ---

insertS :: Ord a => a -> Stream a -> Stream a
insertS a  = apo phi
             where phi s | (headS s) < a  = St (headS s) (i1 (tailS s))
                         | otherwise      = St a (i2 s)

maphd :: (a -> a) -> Stream a -> Stream a
maphd h = apo phi
       where phi x = St (h (headS x)) (i2 (tailS x))

-- (6) Exemplos de Futumorfismo ---

exch :: Stream a -> Stream a
exch = futu phi
       where phi x = St (headS (tailS x))
(consF (St (headS x) (lastF (tailS x))))
```

```
-- visualizador para listas do Haskell

toHList :: Stream a -> [a]
toHList s = [headS s] ++ (toHList (tailS s))
```

Apêndice B

Animador para Cálculos de Processos

```
module ProcType where

--cálculo de processos

import Prelude hiding (filter)

import BasicLib

-- Functores

newtype Nu f = Fin (f (Nu f))

unFin :: Nu f -> f (Nu f) unFin (Fin x) = x

ana :: Functor f => (c -> f c) -> c -> Nu f
ana phi = Fin . fmap (ana phi) . phi


-- Acções (álgebra de sincronização para Ccs)

data Act l = A l | AC l | Nop | Tau | Id  deriving Show

instance (Eq a) => Eq (Act a) where
 x == y                  = eqAct x y
```

```
instance (Ord a) => Ord (Act a) where
 x <= y                  = leqAct x y

prodAct :: (Eq a) => Act a -> Act a -> Act a
prodAct Nop _ = Nop
prodAct _ Nop = Nop
prodAct Id x = x
prodAct x Id = x
prodAct (A x) (AC y) = if (x == y) then Tau else Nop
prodAct (AC x) (A y) = if (x == y) then Tau else Nop
prodAct a1 a2 = Nop

eqAct :: (Eq a) => Act a -> Act a -> Bool
eqAct (A a1) (A a2) = (a1 == a2)
eqAct (AC a1) (AC a2) = (a1 == a2)
eqAct Nop Nop = True
eqAct Tau Tau = True
eqAct Id Id = True

leqAct :: (Ord a) => Act a -> Act a -> Bool
leqAct (A a1) (A a2) = (a1 <= a2)
leqAct (AC a1) (AC a2) = (a1 <= a2)
leqAct Nop _ = True
leqAct Tau _ = True
leqAct Id _ = True

-- Acções (álgebra de sincronização para Csp)

data Act l = A l | Nop | Id  deriving Show

instance (Eq a) => Eq (Act a) where
 x == y                  = eqAct x y

instance (Ord a) => Ord (Act a) where
 x <= y                  = leqAct x y

prodAct :: (Eq a) => Act a -> Act a -> Act a
prodAct Nop _ = Nop
prodAct _ Nop = Nop
prodAct Id x = x
prodAct x Id = x
prodAct (A x) (A y) = if x==y then (A x) else Nop
```

```
eqAct :: (Eq a) => Act a -> Act a -> Bool
eqAct (A a1) (A a2) = (a1 == a2)
eqAct Nop Nop = True
eqAct Id Id = True

leqAct :: (Ord a) => Act a -> Act a -> Bool
leqAct (A a1) (A a2) = (a1 <= a2)
leqAct Nop _ = True
leqAct Id _ = True

-- Acções (uma outra álgebra de sincronização --- co-ocorrência)

data Act l = A l | P (l,l)| Nop | Id  deriving Show

instance (Eq a) => Eq (Act a) where
 x == y                  = eqAct x y

instance (Ord a) => Ord (Act a) where
 x <= y                  = leqAct x y

prodAct :: (Eq a) => Act a -> Act a -> Act a
prodAct Nop _ = Nop
prodAct _ Nop = Nop
prodAct Id x = x
prodAct x Id = x
prodAct (A x) (A y) = P (x,y)
prodAct a1 a2 = Nop

eqAct :: (Eq a) => Act a -> Act a -> Bool
eqAct (A a1) (A a2) = (a1 == a2)
eqAct Nop Nop = True
eqAct Id Id = True
eqAct (P (a,b)) (P (x,y)) = (and[a==x,b==y])

leqAct :: (Ord a) => Act a -> Act a -> Bool
leqAct (A a1) (A a2) = (a1 <= a2)
leqAct Nop _ = True
leqAct Id _ = True
leqAct (P (a,b)) (P (x,y)) = and [a<=x,b<=y]

-- Processos

data Pr a x =  C [(a, x)]  deriving Show
```

```
obsProc :: Pr a x -> [(a, x)
obsProc p = f
            where  (C f) = p

instance Functor (Pr a)
where fmap f (C s) = C (map (id >< f) s)

type Proc a = Nu (Pr a)

-- Combinadores Dinâmicos

nilP :: Proc (Act a) nilP =  Fin (C [])

preP :: Act a -> Proc (Act a) -> Proc (Act a)
preP act p = Fin (C [(act,p)])

sumP ::  Proc (Act a) -> Proc (Act a) -> Proc (Act a)
sumP p q  = Fin (C (pp ++ qq)) where
            (C pp) = (unFin p)
            (C qq) = (unFin q)

-- Combinadores Estáticos (recursivos)

rename :: ((Act a) -> (Act a)) -> Proc (Act a) -> Proc (Act a)
rename f p = ana phi p
             where phi p = C (map (f >< id) (obsP p))


interleaving :: (Proc (Act a), Proc (Act a)) -> Proc (Act a)
interleaving (p,q) = ana alphai (p,q)

restriction :: (Eq a) => (Proc (Act a), [Act a]) -> Proc (Act a)
restriction (p,la) = ana  phi p
                     where phi p = C (filter (obsP p) la)

-- Produto

syn :: (Eq a) => (Proc (Act a),Proc (Act a)) -> Proc (Act a)
syn (p, q) = ana alphap (p, q)

-- Paralelo
```

```
par :: (Eq a) => (Proc (Act a), Proc (Act a)) -> Proc (Act a)
par (p, q) = ana alpha (p, q)
             where alpha (p, q) =
         C ((obsProc (alphai (p,q))) ++ (obsProc (alphap (p,q))))

-- Funções Auxiliares

obsP :: Proc (Act a) -> [(Act a, Proc (Act a))]
obsP p = pp
         where (C pp) = (unFin p)

taur ::  [(a,c)] -> c ->  [(a,(c,c))]
taur s x =  [((p1 e),((p2 e),x))|e <-  s]

taul ::  c -> [(a,c)] -> [(a,(c,c))]
taul x s =  [((p1 e),(x, (p2 e)))|e <-  s]

filter ::  (Eq a) => [(a,b)] -> [a] -> [(a,b)]
filter k l = [ x | x <- k,(notElem (p1 x) l) == True]

filtro p la = C (filter (obsP p) la)

alphai :: (Proc (Act a),Proc (Act a)) ->
      Pr (Act a) (Proc (Act a),Proc (Act a))
alphai (p, q) =  C ((taur (obsP p) q) ++ (taul p (obsP q)))

alphap :: (Eq a) =>(Proc (Act a),Proc (Act a)) ->
                   Pr (Act a) (Proc (Act a),Proc (Act a))
alphap (p, q) = C (sel (delta (obsP p) (obsP q)))

delta ::  (Eq a) => [((Act a),c)] -> [((Act a),c)] -> [((Act a),(c,c))]
delta r s = [((prodAct (p1 e1) (p1 e2)), ((p2 e1), (p2 e2)))
            | e1 <- r, e2 <- s]

sel :: Eq (Act a) => [((Act a ), b)] -> [(Act a, b)]
sel p = filter p [Nop]

-- Visualizador de Processos

data ShowProc a  =  V [(a, ShowProc a)]  deriving Show

vP :: Proc a -> ShowProc a
vP (Fin (C l)) = V (vPaux l)
```

```
vPaux :: [(a, Proc a)] -> [(a, ShowProc a)]
vPaux [] = []
vPaux ((a,p):t) =  [(a, (vP p))] ++ (vPaux t)

-- Executa processo

execProc :: (Show a) => Proc (Act a) -> IO ()
execProc (Fin (C l)) =
   case (length l) of
      0 -> putStrLn "Processo terminado"
      1 -> do putStrLn
        ("Accao " ++ show (fst . head $ l) ++ " executada.")
                execProc (snd . head $ l)
      _ -> do putStrLn .
        ((++) "Escolha a posicao da accao a executar")
                . show $ (map fst l)
                i <- getLine
                let p = posi (toInteiro i) l
                putStrLn ("Accao " ++ show (fst p) ++ " executada.")
                execProc (snd p)



toInteiro :: String -> Int
toInteiro = read

posi i = last . take i
```

Referências

[AHS90] J. Adamek, H. Herrlich, and G. E. Strecker. *Abstract and Concrete Categories*. John Wiley & Sons, Inc (revised electronic edition in 2004), 1990.

[AM88] P. Aczel and N. Mendler. A final coalgebra theorem. In D. Pitt, D. Rydeheard, P. Dybjer, A. Pitts, and A. Poigne, editors, *Proc. Category Theory and Computer Science*, pages 357–365. Springer Lect. Notes Comp. Sci. (389), 1988.

[Bac03] R. Backhouse. *Program Construction*. John Wiley and Sons, Inc., 2003.

[Bal00] M. Baldamus. Compositional constructor interpretation over coalgebraic models for the π-calculus. In H. Reichel, editor, *Proc. of CMCS'00*, volume 33. Elect. Notes in Theor. Comp. Sci., Elsevier, 2000.

[Bar01a] L. S. Barbosa. *Components as Coalgebras*. PhD thesis, DI, Universidade do Minho, 2001.

[Bar01b] L. S. Barbosa. Process calculi *à la* Bird-Meertens. In *CMCS'01*, volume 44.4, pages 47–66, Genova, April 2001. Elect. Notes in Theor. Comp. Sci., Elsevier.

[Bar01c] F. Bartels. Generalised coinduction. In Andrea Corradini, Marina Lenisa, and Ugo Montanari, editors, *CMCS'01, Elect. Notes in Theor. Comp. Sci.*, volume 44.4, pages 67–87, Genova, April 2001. Elsevier.

[BH93] R. C. Backhouse and P. F. Hoogendijk. Elements of a relational theory of datatypes. In B. Möller, H. Partsch, and S. Schuman, editors, *Formal Program Development*, pages 7–42. Springer Lect. Notes Comp. Sci. (755), 1993.

[Bir87] R. S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, volume 36 of *NATO ASI Series F*, pages 3–42. Springer-Verlag, 1987.

[Bir98] R. Bird. *Functional Programming Using Haskell*. Series in Computer Science. Prentice-Hall International, 1998.

[BM87] R. S. Bird and L. Meertens. Two exercises found in a book on algorithmics. In L. Meertens, editor, *Program Specification and Transformation*, pages 451–458. North-Holland, 1987.

[BM97] R. Bird and O. Moor. *The Algebra of Programming*. Series in Computer Science. Prentice-Hall International, 1997.

[BO02] L. S. Barbosa and J. N. Oliveira. Coinductive interpreters for process calculi. In *Proc. of FLOPS'02*, pages 183–197, Aizu, Japan, September 2002. Springer Lect. Notes Comp. Sci. (2441).

[Bor94] F. Borceux. *Handbook of Categorial Algebra (3 volumes)*. Cambridge University Press, 1994.

[BW85] M. Barr and C. Wells. *Toposes, Triples and Theories*. Springer-Verlag, 1985.

[BW90] M. Barr and C. Wells. *Category Theory for Computer Scientists*. Series in Computer Science. Prentice-Hall International, 1990.

[dB80] J. de Bakker. *Mathematical Theory of Program Correctness*. Prentice Hall, 1980.

[DS90] E. W. Dijkstra and C. S. Scholten. *Predicate Calculus and Program Semantics*. Springer Verlag, NY, 1990.

[GH05] J. Gibbons and G. Hutton. Proof methods for corecursive programs. *Fundamenta Informatica*, 66(4):353–366, 2005.

[Gib97] J. Gibbons. Conditionals in distributive categories. CMS-TR-97-01, School of Computing and Mathematical Sciences, Oxford Brookes University, 1997.

[Gib02] J. Gibbons. Algebraic and coalgebraic methods for calculating functional programs. In R. Crole, R. Backhouse, and J. Gibbons, editors, *Algebraic and Coalgebraic Methods in the Mathematics of Program Constuction*, pages 236–280. Springer Lect. Notes Comp. Sci. (2297), 2002.

[GS93] D. Gries and F. Schneider. *A Logical Approach to Discrete Mathematics*. Springer Verlag, NY, 1993.

[Gum99] H. Peter Gumm. Elements of the general theory of coalgebras. Technical report, Lecture Notes for LUTACS'99, South Africa, 1999.

[Hag87] T. Hagino. A typed lambda calculus with categorical type constructors. In D. H. Pitt, A. Poigné, and D. E. Rydeheard, editors, *Category Theory and Computer Science*, pages 140–157. Springer Lect. Notes Comp. Sci. (283), 1987.

[HC84] G. E. Hughes and M. J. Cresswell. *A Companion to Modal Logic*. Metheuen (UP), 1984.

[Hoa85] C. A. R Hoare. *Communicating Sequential Processes*. Series in Computer Science. Prentice-Hall International, 1985.

[Jac99] B. Jacobs. The temporal logic of coalgebras via Galois algebras. Techn. rep. CSI-R9906, Comp. Sci. Inst., University of Nijmegen, 1999.

[JR97] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–159, 1997.

[Kel82] G. M. Kelly. *Basic Concepts of Enriched Category Theory*, volume 64 of *London Mathematical Society Lecture Notes Series*. Cambridge University Press, 1982.

[Koc72] A. Kock. Strong functors and monoidal monads. *Archiv für Mathematik*, 23:113–120, 1972.

[Kur01] A. Kurz. *Logics for Coalgebras and Applications to Computer Science*. Ph.D. Thesis, Fakultat fur Mathematik, Ludwig-Maximilians Univ., Muenchen, 2001.

[Len98] M. Lenisa. *Themes in Final Semantics*. PhD thesis, Universita de Pisa-Udine, 1998.

[Mac71] S. Mac Lane. *Categories for the Working Mathematician*. Springer Verlag, 1971.

[Mal90] G. R. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14(2–3):255–279, 1990.

[MB04] Sun Meng and L. S. Barbosa. On refinement of generic software components. In C. Rettray, S. Maharaj, and C. Shankland, editors, *10th Int. Conf. Algebraic Methods and Software Technology (AMAST)*, pages 506–520, Stirling, 2004. Springer Lect. Notes Comp. Sci. (3116). Best Student Co-authored Paper Award.

[McL92] C. McLarty. *Elementary Categories, Elementary Toposes*, volume 21 of *Oxford Logic Guides*. Clarendon Press, 1992.

[MFP91] E. Meijer, M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In J. Hughes, editor, *Proceedings of the 1991 ACM Conference on Functional Programming Languages and Computer Architecture*, pages 124–144. Springer Lect. Notes Comp. Sci. (523), 1991.

[Mil80] R. Milner. *A Calculus of Communicating Systems*. Springer Lect. Notes Comp. Sci. (92), 1980.

[Mil89] R. Milner. *Communication and Concurrency*. Series in Computer Science. Prentice-Hall International, 1989.

[Mil99] R. Milner. *Communicating and Mobile Processes: the π-Calculus*. Cambridge University Press, 1999.

[Mog91] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[Mos99] L. Moss. Coalgebraic logic. *Ann. Pure & Appl. Logic*, 1999.

[Par81] D. Park. Concurrency and automata on infinite sequences. pages 561–572. Springer Lect. Notes Comp. Sci. (104), 1981.

[Pol45] G. Polya. *How to Solve It: A New Aspect of Mathematical Method*. Princeton University Press, 1945.

[RM02] J. Rothe and D. Masulovic. Towards weak bisimulation for coalgebras. In A. Kurz, editor, *Proc. Int. Workshop on Categorical Methods for Concurrency, Interaction, and Mobility (CMCIM'02)*, volume 68. Elect. Notes in Theor. Comp. Sci., Elsevier, 2002.

[Rut00] J. Rutten. Universal coalgebra: A theory of systems. *Theor. Comp. Sci.*, 249(1):3–80, 2000. (Revised version of CWI Techn. Rep. CS-R9652, 1996).

[Rut01] J. Rutten. Elements of stream calculus (an extensive exercise in coinduction). Technical report, CWI, Amsterdam, 2001.

[Rut05] J. Rutten. A tutorial on coinductive stream calculus and signal flow graphs. *Theor. Comp. Sci.*, 343(3):443–481, 2005. (An extended abstract of this paper appeared in LNCS 3188, 2004).

[Sch98] D. Schamschurko. Modeling process calculi with PVS. In *CMCS'98, Elect. Notes in Theor. Comp. Sci.*, volume 11. Elsevier, 1998.

[SVW05] A. Sokolova, E. de Vink, and H. Woracek. Weak bisimulation for action-type coalgebras. In L. Birkedal, editor, *Proc. Int. Conf. on Category Theory and Computer Science (CTCS'04)*, volume 122, pages 211–228. Elect. Notes in Theor. Comp. Sci., Elsevier, 2005.

[UV99] T. Uustalu and V. Vene. Primitive (co)recursion and course-of-values (co)iteration, categorically. *INFORMATICA (IMI, Lithuania)*, 10(1):5–26, 1999.

[Ven00] V. Vene. *Categorical Programming with Inductive and Coinductive Types*. PhD thesis, Faculty of Mathematics, University of Tartu (*Dissertationsa Mathematicae* 23), 2000.

[VU97] V. Vene and T. Uustalu. Functional programming with apomorphisms (corecursion). In *Proc. 9th Nordic Workshop on Programming Theory*, 1997.

[Wol99] U. Wolter. A coalgebraic introduction to CSP. In *Proc. of CMCS'99*, volume 19. Elect. Notes in Theor. Comp. Sci., Elsevier, 1999.

[Zei99] P. Zeitz. *The Art and Craft of Problem Solving*. John Wiley and Sons, Inc., 1999.